Digital Video Broadcasting (DVB); Implementation guidelines for Data Broadcasting

DVB Document A047
June 1999

DVB Project Office
7th June 1999

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 SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available **free of charge** from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

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 SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

1 Scope

The present document provides implementation guidelines for the use and implementation of the Digital Video Broadcasting (DVB) data broadcast service (see EN 301 192 [1]) in a DVB digital broadcast environment including satellite networks cable networks MMDS networks and terrestrial networks.

The present document gives highly recommended rules for the usage of EN 301 192 [1]. As such, it facilitates the efficient and reliable implementation of data broadcast services. The rules apply to broadcasters, network operators as well as manufacturers.

The rules are specified in the form of constraints on the data broadcast implementation.

The specification of these functions in no way prohibits end consumer device manufacturers from including additional features, and should not be interpreted as stipulating any form of upper limit to the performance.

The present document uses the terminology defined in EN 301 192 [1] and should be read in conjunction with that document.

NOTE: It is highly recommended that the end consumer device should be designed to allow for future compatible extensions to the DVB data broadcast specification EN 301 192 [1]. All the fields "reserved" (for ISO), "reserved_future_use" (for ETSI), and "user defined" in EN 301 192 [1] should be ignored by end consumer devices not to make use of them. The "reserved" and "reserved_future_use" field may be specified in the future by the respective bodies, whereas the "user defined" field will not be standardized.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] EN 301 192: "Digital Video Broadcasting (DVB); DVB specification for data broadcasting".

[2] ETS 300 802: "Digital Video Broadcasting (DVB); Network-independent protocols for DVB interactive services".

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

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

[5] RFC 791 (IP) (1981): "Internet Protocol", J. Postel.

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

[7] EN 300 472: "Digital Video Broadcasting (DVB); Specification for conveying ITU-R System B Teletext in DVB bitstreams".

[8] ETS 300 743: "Digital Video Broadcasting (DVB); Subtitling systems".

[9] Common Object Request Broker (1995): "Architecture and Specification OMG", Revision 2.0.

[10] RFC 1521 (1993): "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", N. Borenstein, N. Freed.

[11] RFC 1590 (1994): "Media Type Registration Procedure", (Updates RFC 1521), J. Postel.

[12] OMT (1995): "The Object Model, JOOP", James Rumbaugh.

[13] ISO/IEC 8802: "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements".

[14] RFC 1951

[15] RFC 1112 (1989): "Host extensions for IP multicasting", S.E. Deering.

3 Definitions and abbreviations

3.1 Definitions

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

Broadcaster (SERVICE Provider): an organization which assembles a sequence of events or programmes to be delivered to the viewer based upon a schedule.

Component (ELEMENTARY Stream): one or more entities which together make up an event, for example video, audio, teletext, data.

DSM-CC: refers to ISO/IEC 13818-6 [4]. Digital Storage Media - Command & Control is defined in part 6.

LLC/SNAP: refers to ISO/IEC 8802 [13] Logical Link Control (part 2) and SubNetwork Attachment Point (part 1a).

MPEG2: refers to ISO/IEC 13818-1 [3]. Systems coding is defined in part 1. Video coding is defined in part 2. Audio coding is defined in part 3.

multiplex: a stream of all the digital data carrying one or more services within a single physical channel.

section: a section is a syntactic structure used for mapping all service information into ISO/IEC 13818-1 [3] Transport Stream packets.

Service Information (SI): digital data describing the delivery system, content and scheduling/timing of broadcast data streams etc. It includes MPEG2 Program Specific Information (PSI) together with independently defined extensions.

sub-table: a sub-table is comprised of a number of sections with the same value of table_id, table_id_extension and version_number. The table_id_extension field is equivalent to the fourth and fifth byte of a section when the section_syntax_indicator is set to a value of "1".

table: a table is comprised of a number of sections with the same value of table_id.

Transport Stream (TS): a data structure defined in ISO 13818-1 [3]. It is the basis of the DVB standards.

3.2 Abbreviations

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

API	Application Portability Interface
BIOP	Broadcast Inter ORB Protocol
bslbf	bit string, left bit first
CDR	Common Data Representation
CRC	Cyclic Redundancy Check
CORBA	Common Object Request Broker Architecture
DDB	DownloadDataBlock message of DSM-CC

DII	DownloadInfoIndication message of DSM-CC
DSI	DownloadServerInitiate message of DSM-CC
DSM-CC	Digital Storage Media - Command & Control
DSM-CC U-N	DSM-CC User to Network
DSM-CC U-U	DSM-CC User to User
DVB	Digital Video Broadcasting
EIT	Event Information Table
GIF	Graphics Interchange Format
HTML	HyperText Mark-up Language
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IIOP	Internet Inter ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
JPEG	Joint Photographic Experts Group
LLC	Logical Link Control
MAC	Medium Access Control
MPEG	Moving Pictures Expert Group
MTU	Maximum Transport Unit
NPT	Normal Play Time
NSAP	Network Service Access Point
OMG	Object Management Group
OMT	Object Modelling Technique
ORB	Object Request Broker
PAT	Program Association Table
PCR	Program Clock Reference
PES	Packetized Elementary Stream
PID	Packet Identifier
PLL	Phase Locked Loop
PMT	Program Map Table
PPP	Point to Point Protocol
ppm	parts per million
PSI	Program Specific Information
PTS	Presentation Time Stamp
RFC	Request For Comments
SDT	Service Description Table
SI	Service Information
SIS	Systems for Interactive Services
SNAP	SubNetwork Attachment Point
TCP	Transport Control Protocol
TS	Transport Stream
uimsbf	unsigned integer, most significant bit first

4 Rules of operation

This Clause contains recommendations on the usage of the DVB data broadcasting specification EN 301 192 [1].

4.1 Introduction

Figure 4.1 gives an overview of the data broadcast specification EN 301 192 [1].

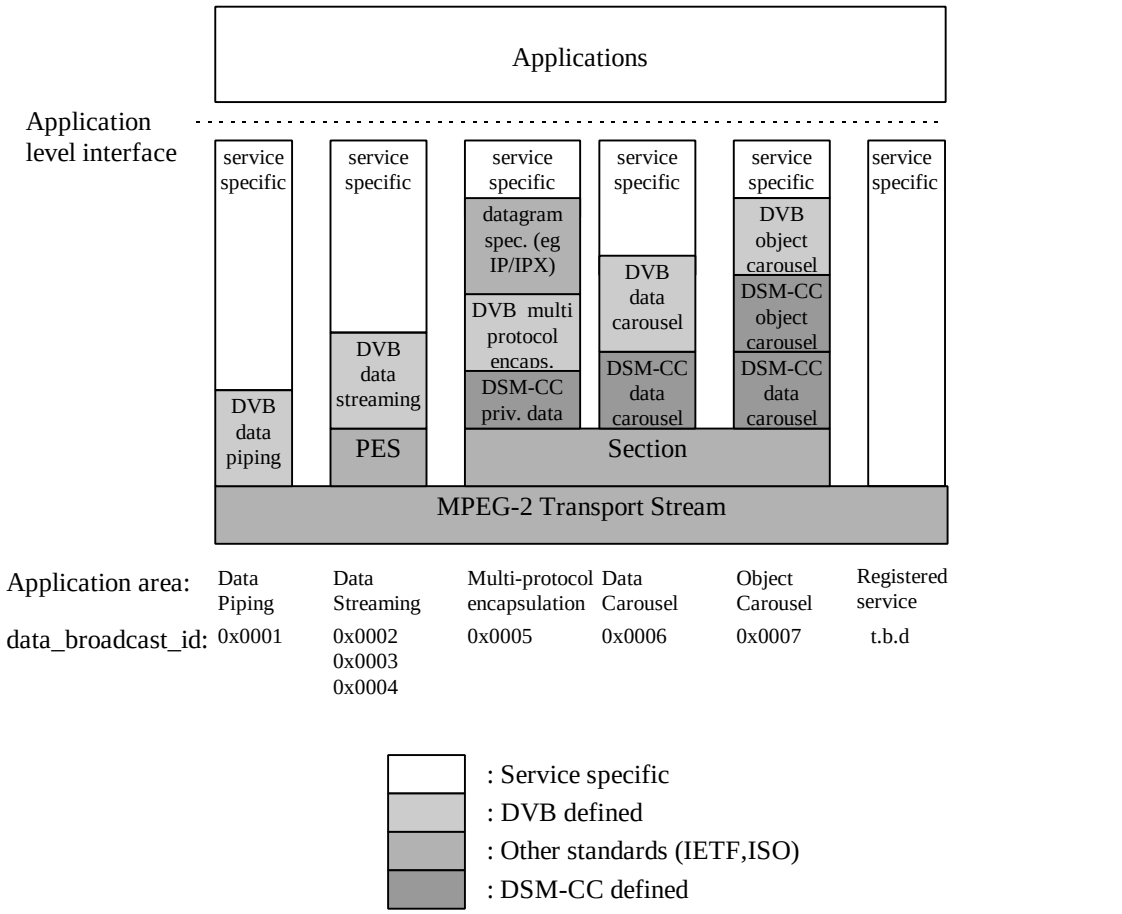


Figure 4.1: Graphical overview and relation to other standards

The basis of the complete specification EN 301 192 [1] is formed by the MPEG2 Transport Stream (TS) as defined in ISO/IEC 13818-1 [3] (MPEG2 Systems). Data information can be transported within this MPEG2 TS by means of application areas. The application areas are:

- Data piping;
- Data streaming;
- Multiprotocol encapsulation;
- Data Carousel.

Furthermore in Figure 4.1 the object carousel is depicted. This carousel is used by the specification for Network Independent Protocols for Interactive Services ETS 300 802 [2].

A registered service is shown on the right hand side of the figure. DVB allows to register private implementations for data broadcast services, as described in EN 301 192 [1] annex A.

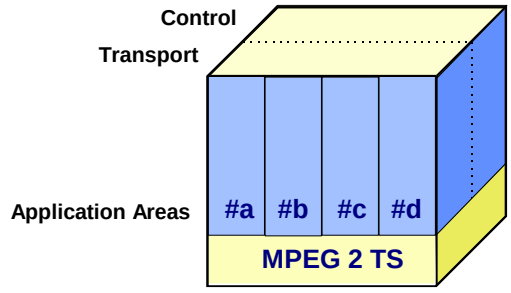
Figure 4.1 shows what is standardized by which body. ISO has standardized the MPEG2 TS in ISO/IEC 13818-1 [3] and the DSM-CC framework in ISO/IEC 13818-6 [4]. IETF has standardized the Internet Protocol (IP) in RFC 791 [5]. DVB has specified within the data broadcast specification EN 301 192 [1] the DVB data piping, DVB data streaming, DVB multiprotocol encapsulation, DVB Data Carousel and DVB object carousel parts. Within Figure 4.1 the encapsulation of the Internet Protocol (IP) is just an example. Other protocols can also be encapsulated.

As shown in Figure 4.1, the specification for data broadcast EN 301 192 [1] specifies different service levels for all application areas. The data piping specification does not give much information on how to get the data out of the MPEG2 TS. It more or less only specifies how to put data into MPEG2 Transport Stream packets. In comparison with the other application areas a lot of service specific hard - and/or software has to be built to get a service running.

The data streaming specification gives some more functionality, especially for timing. It is possible to do asynchronous data broadcast, synchronized broadcast or synchronous broadcast. The specification EN 301 192 [1] is based on PES packets as defined in MPEG2 ISO/IEC 13818-1 [3].

The multiprotocol encapsulation, Data Carousel and object carousel application areas specifications are all built using the DSM-CC framework of MPEG2 (ISO/IEC 13818-6 [4]). It is based on MPEG2 private sections as defined in MPEG2 ISO/IEC 13818-1 [3]. DVB has added specific information to get the framework working in the DVB environment, especially in conjunction with the Service Information specification EN 300 468 [6].

In the specification for data broadcasting EN 301 192 [1], every application area is defined by two parts as shown in Figure 4.2.



Control: SI and PSI
Transport: Databroadcast transport specification

Figure 4.2: Transport and Control specification parts

The control specification is part of the Service Information specification EN 300 468 [6], the transport part of the specification is part of the data broadcast specification EN 301 192 [1].

The following subclauses give implementation guidelines how to use the different application areas.

4.2 Selection of the Appropriate Profile

As shown in Figure 4.1 there are different ways to transmit data via MPEG2 DVB Transport Streams. The mechanisms have different characteristics concerning filtering, overhead, size etc. The selection of the appropriate mechanism has to be based on the specific requirements of the target application.

The level of detail of the specification varies for the different profiles. In case of Multiprotocol Encapsulation (see EN 301 192 [1] Clause 6) and Data Carousels (see EN 301 192 [1] Clause 6) the specification is very detailed, which only requires very few application specific definitions, in case of the other profiles there is much freedom.

Furthermore, it is possible to use application areas for other purposes than the recommended ones; for example a Data Carousel like application can be based on top of Data piping and an IP broadcast one on top of Data streaming. Such arrangements are of course part of service specific choices.

4.2.1 Fragmentation of Datagrams

Generally data of any kind of protocols are transmitted in packetized form ("datagrams"). These datagrams may have different length. If the data are not packetized or the packetization method is irrelevant or hidden to the DVB transmission chain the most appropriate way of transmission is the Data Pipe (see EN 301 192 [1] Clause 3).

On the layer of MPEG2 Transport Stream data are transmitted within packets with a fixed length of 188 bytes (184 bytes payload), therefore datagrams of higher layers have to be fragmented at the transmission side and be re-assembled at the reception. For fragmentation of the datagrams there are three possible ways (see also Figure 4.1):

- private mechanisms based on the Data Pipe;
- MPEG2 Packetized Elementary Streams (PES);
- MPEG2 Sections.

MPEG2 PES provides a mechanism to transmit datagrams of variable size with a maximum length of 64 Kbytes. Additionally it provides the facility to synchronize different data streams accurately (as used in MPEG for synchronization of Video and Audio), therefore it was chosen by DVB for the transmission of synchronous and synchronized but also asynchronous data streams (see EN 301 192 [1], Clause 4 and Clause 5).

MPEG2 Sections can be used to transmit datagrams of variable size with a maximum length of 4 Kbytes. The transmission is asynchronous. MPEG2 Sections are built in a way that MPEG2 demultiplexers available on the market can filter out single sections in hardware which may reduce the required software processing power of the receiver. This is the main reason why the MPEG2 Sections have been chosen as the mechanism for the transmission of encapsulated protocols and Data Carousels.

For data broadcasting services in the DVB framework the data_broadcast_id_descriptor EN 300 468 [6] can be present in the PMT, i.e. use of this descriptor is optional.

4.3 Data Pipe

The Data Pipe is an asynchronous transportation mechanism for data. The data are inserted directly in the payload of MPEG2 Transport Packets.

There is no mechanism for fragmentation and reassembly of datagrams defined. This, if required, is part of the application definition. For instance, the payload_unit_start_indicator could be used to signal the start of a datagram in a packet while the transport_priority field could signal the end of a datagram.

The continuity_counter shall be used as defined by MPEG (ISO/IEC 13818-1 [3], Subclause 2.4.3).

4.3.1 Usage of the Adaptation Field

The main use of the Adaptation Field is stuffing. However, it is possible to use it for other purposes, for example the transmission of PCR.

4.4 Asynchronous/Synchronized/Synchronous Data Streaming

4.4.1 Usage of the Adaptation Field

According to ISO/IEC 13818-1 [3], Subclause 2.4.3 a PES packet always has to start at the first payload byte of an MPEG2 Transport Packet. This means that for PES packets which are not aligned with the MPEG2 Transport Packet there is stuffing necessary. Since MPEG only allows stuffing bytes at the end of the packet for sections and not for PES (see ISO/IEC 13818-1 [3], Subclause 2.4.3.3) stuffing can only be achieved by using Adaptation fields. This is no real constraint for the performance of a decoder since most demultiplexer implementations provide the automatic extraction of Adaptation Fields and therefore no additional processing power is required.

An Adaptation Field that is only used for stuffing can be created by setting all adaptation field flags (discontinuity_indicator, random_access_indicator, elementary_stream_priority_indicator, PCR_flag, OPCR_flag, splicing_point_flag, transport_private_data_flag, adaptation_field_extension_flag) to "0" and inserting the number of required stuffing bytes.

The elementary_stream_priority_indicator and the adaptation_field_extension_flag shall be set to zero, since the affiliated features have no meaning for Data Streaming.

4.4.2 Asynchronous Data Streaming

Asynchronous Data Streaming is used in the case that the PES mechanism is of advantage for applications that need the asynchronous transmission of datagrams.

Since no synchronization is necessary for this kind of transmission the stream_id "private_stream_2" (see ISO/IEC 13818-1 [3]) has been chosen which implicitly excludes the usage of the PES packet header fields. Therefore the PES_packet_length field is immediately followed by the datagram.

The definition of the datagram format is part of the private implementation ant therefore not subject of the DVB specification.

4.4.3 Synchronous/Synchronized Data Streaming

In order to meet the requirements of the Synchronous and Synchronized Data Streaming an additional header, specific to this DVB application profile has been defined (see EN 301 192 [1], Table 5.1).

The field stream_id shall be set to "private_stream_1", allowing for the usage of the PES header fields, especially the PTS. Usage of the time stamps requires the definition of Access Units. Since this is application dependant it has not been defined within the DVB data broadcasting specification.

The first byte of this header (which is from MPEG2 PES point of view the first payload byte) contains the data_identifier. It is defined in accordance with the specifications for embedding of EBU-data (EN 300 472 [7]) and DVB-subtitling (ETS 300 743 [8]) and indicates the type of Data Streaming (synchronous /synchronized).

A combination of Synchronized and Synchronous Data Streaming in the same PES packet is not allowed. However, both types of streaming data can be carried as part of a same program in separate PIDs.

The field sub_stream_id may be used for private definition.

The two flags PTS_extension_flag and output_data_rate_flag indicate the existence of an output data rate field and of a field for PTS extension. The usage of these fields is explained below.

The PES_data_packet_header_length indicates the length of the header and allows the addition of private bytes in the header.

The DTS field in PES header is of no use while the PTS shall be coded in the same way as defined by MPEG ISO/IEC 13818-1 [3].

4.4.4 Synchronous Data Streaming

Synchronous data streaming may be used if the output data rate at the receiver side needs to be very accurate. The receiver clock is synchronized by the PCR. The 9-bit PTS_extension is needed to position data access units (a bit, a byte or few bytes depending on how one defines access units) very accurately over a large range of data rates (kbit/sec to Mbit/s). The 9 bits extends the accuracy of the PTS clock from 11 microseconds to the same accuracy as a 27 MHz clock (37 nanoseconds). Precise positioning of the data is required if multiple data decoders receiving the same data services (and knowing the latency through the process) have to output the data at the same time in an aligned way, or if it is required to maintain synchronization in the data output stream following a temporary loss of signal.

The field output_data_rate is used in order to specify the output data rate for the synchronous data stream. With the 28-bit accuracy (instead of the 400 bit/s resolution of 22-bit ES_rate in PES header) it is possible to implement PLL (with clock down conversion) with a ratio of data output rate to 27 MHz (± 30 ppm) covering a wide range of data rates. The output_data_rate field conveys the bit rate of the regenerated signal for a synchronous data stream. The encoding of the bit rate of the data stream into the output_data_rate field depends on the application. Applications which require bit rates which are a multiple of 1 bit/s may encode the data streams bit rate into the output_data_rate field directly with the units of output_data_rate as bits/second. Applications which require a continuous range of bit rates to be regenerated within the 30 ppm tolerance specified by MPEG for the 27 MHz system_clock_frequency may encode the bit rate of the data stream into the output_data_rate field as:

output_data_rate = bit rate ´ M/system_clock_frequency

where M is chosen to be a number sufficiently large to express the range of bit rates required for the application with the desired bit rate accuracy. The practical range of bit rates for synchronous data streaming with a system_clock_frequency of 27 MHz is 1 bit/s to 27 Mbit/s.

NOTE: The decoder model described in Clause 10 of EN 301 192 [1] is not necessarily applicable if the output data rate field is used.

ES_rate in the PES header can be used without the output_data_rate field in the PES data_packet for applications where the 400 bit/s accuracy of ES_rate is adequate (for example T1 and E1). If both ES_rate and output_data_rate are present in an encoded stream, the decoder can use either of the rates.

The recommended buffer size for synchronous data streaming is 4 800 byte. This gives sufficient capacity for a typical maximum multiplexing jitter of 4 ms and a bit rate up to 9 Mbit/s.

4.4.5 Synchronized Data Streaming

Synchronized Data Streaming is used when the data stream shall be synchronized with another MPEG2 PES stream.

4.5 Multiprotocol encapsulation

4.5.1 Overview

The multiprotocol encapsulation provides a mechanism for transporting data network protocols on top of the MPEG2 Transport Streams in DVB networks. It has been optimized for carriage of the Internet Protocol (IP) (RFC 791 [5]), but can be used for transportation of any other network protocol by using the LLC/SNAP encapsulation. It covers unicast (datagrams targeted to a single receiver), multicast (datagrams targeted to a group of receivers) and broadcast (datagrams targeted to all receivers). 48 bit MAC addresses are used for addressing receivers. However, DVB does not specify how the MAC addresses are allocated to the receivers.

Due to the broadcast nature of DVB networks, security of the data is very important. The encapsulation allows secure transmission of data by supporting encryption of the packets and dynamically changing MAC addresses.

4.5.2 Data transport

The datagrams are transported in datagram_sections which are compliant to the DSMCC_section format for private data. The section format provides an efficient format for mapping the datagrams to the MPEG2 Transport Stream packets and support filtering of datagrams based on the MAC address using existing hardware or software demultiplexers.

The section format permits fragmenting datagrams into multiple sections. If the length of the datagram is less or equal than 4 080 bytes (including the possible LLC/SNAP header), the datagram shall be sent in one section. In case of IP and the LLC/SNAP header being omitted, the MTU shall be set to 4 080 bytes or less, so that the datagrams will never be fragmented. In case of IP and the LLC/SNAP header being present the MTU shall be set to 4 074 or less.

The MAC address has been divided into 6 bytes that are located in two groups. The reason for this is that the bytes 5 and 6 are in place of the table_id_extension field of the DSMCC_section while bytes 1,2,3 and 4 are in the payload area of the DSMCC_section.

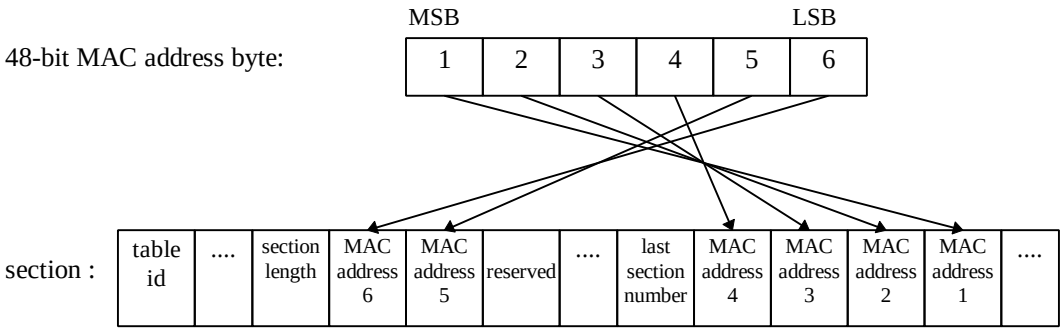


Figure 4.2a

Some demultiplexers are able to filter bytes 5 and 6 with hardware while filtering bytes 1,2,3 and 4 is done in software. It is recommended that the two bytes of the MAC address which most probably differentiate the receivers are put to the bytes 5 and 6. This is normally the case with IEEE MAC addresses and it is recommended that all MAC addresses are constructed in this way.

Payload scrambling is controlled by a 2-bit field payload_scrambling_control. If the value of these bits is "00", the payload is not scrambled. If the payload is scrambled, the scrambling algorithm is private to the service. The mechanism how the scrambling method is signalled to the receiver is not defined by DVB.

MAC address scrambling provides further security by dynamically changing MAC addresses. By changing the control word that is used for scrambling the MAC addresses periodically, monitoring of the stream can be prevented as the destination of a particular datagram can not be determined by observing the MAC addresses. It also strengthens the security as collecting datagrams destined to a single receiver is difficult. The delivery mechanism of control words used for scrambling the MAC addresses is not defined by DVB.

Address scrambling is controlled in the section header by a 2-bit field address_scrambling_control. If the value is these bits is ‘00’, the MAC address is not scrambled. It should be noted that using MAC address scrambling without payload scrambling is of no use, since the protocol address that is part of the datagram is visible in the clear.

The LLC/SNAP encapsulation provides a multiprotocol encapsulation that can be used for carrying any network protocol, including IP. There is an optimization for carrying IP that allows transmitting IP datagrams without the LLC/SNAP header. This is controlled by the LLC_SNAP_flag bit in the section header. If the value of the bit is "0", the payload contains a bare IP datagram. If the value of the bit is "1", the payload contains an LLC/SNAP encapsulation which consists of the LLC/SNAP structure LLC_SNAP() followed by the datagram bytes. The optimized way of carrying IP can be used for both IPv4 and Ipv6. The section_number and last_section_number fields have to both be "0" when carrying the IP protocol.

The section may contain stuffing after the datagram. The stuffing bytes may be used, for example, for making the payload of the section to be multiple of a block size when a block encryption code is used. The value of these bytes is not specified and in case of payload encryption they should not be assigned a fixed value as it would help breaking the encryption.

The datagram_section has a checksum or a CRC_32 in the end depending on the value of the section_syntax_indicator. It is recommended to use the CRC_32 as it provides a slightly better protection against bit errors as it can be checked by hardware in most hardware demultiplexers while the checksum has to be normally checked by software.

4.5.3 Information in the SI

For services using multiprotocol encapsulation, the data_broadcast_descriptor shall be present in the SDT or the EIT. The multiprotocol_encapsulation_info structure EN 301 192 [1] is carried in the selector_byte field.

MAC_address_range field is used for signalling to the receiver which bytes of the MAC_address are significant for filtering. The significant bytes of the MAC address are at the least significant end of the MAC address.

The MAC_IP_mapping_flag signals if mapping multicast IP addresses to MAC addresses are done according to the RFC 1112 [15]. It should be noted that as DVB does not define that the MAC addresses are used as defined by IEEE, alternative, possibly more optimized, mappings are allowed.

Alignment indicator indicates if the datagram_section is 8-bit aligned or 32-bit aligned to the Transport Stream packets. An 8-bit alignment essentially means that it is not aligned. Alignment is useful in implementations which input Transport Stream packets and rely on the beginning of the section being on a 32-bit boundary for enabling efficient comparison operations in filtering. The alignment is performed using the adaptation field of the Transport Stream packet and/or stuffing bytes at the end of the sections.

The max_sections_per_datagram field defines the maximum number of section that are used for carrying a single datagram (for IP this is restricted to be 1). This defines the maximum length of the datagram. Typically a receiver has to combine the fragments of the datagram before passing it on, so this field defines the size of the buffer that the receiver has to have for combining a datagram of the maximum length.

4.6 Data Carousel

4.6.1 Introduction

The Data Carousel is a transport mechanism that allows a server (the broadcast component of an application) to present a set of distinct data modules to a decoder (a program that is run by a receiver) by cyclically repeating the contents of the carousel, one or more times. If an application decoder wants to access a particular module from the Data Carousel, it may simply wait for the next time that the data for the requested module is broadcast.

A good example of the Data Carousel concept that is widely understood is the Teletext system. In this system, a complete set of Teletext pages is cyclically broadcast in some of the lines of an analogue video signal that are not part of the active picture. When users requests a page, they have to usually wait for the next time the page is broadcast. The maximum length of time the user has to wait can be determined by the time it takes for a complete cycle of the carousel, which in turn can be deduced from the size of the carousel and the rate at which data can be broadcast.

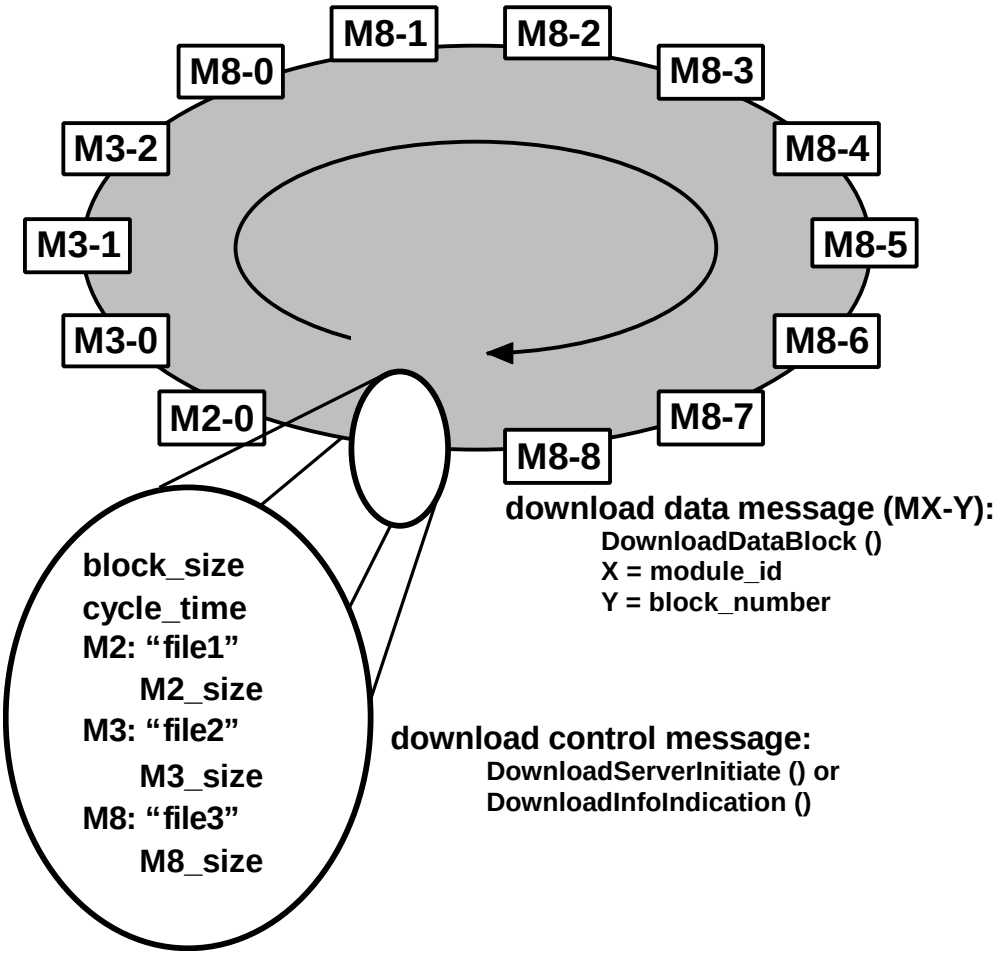


Figure 4.3: Cyclic transmission of information in a Data Carousel

Within a Data Carousel the data is structured into Modules, depicted in Figure 4.3 as M2, M3 and M8. This could simply be the contents of a number of files, say "file1", "file2" and "file3" as in this example. Each Module is divided up to form the payload of one or more **download data messages** each defined using the DSM-CC DownloadDataBlock syntax. The number of such messages depends on the size of the Module and the maximum payload of each download data message. Information describing each Module and any logical grouping is provided by **download control messages**, defined using either the DSM-CC DownloadServerInitiate or DownloadInfoIndication syntaxes as appropriate.

In this example each download message has been inserted only once and DownloadDataBlocks from the same Module have been inserted adjacent to one another and in order. There are however, no restrictions on how often a particular message is inserted into a single loop of the carousel and the order and relative position of messages. This allows the Data Carousel to be created in whatever way best suits a particular use. In addition the frequency and order of insertion of messages into the Data Carousel do not need to be fixed and can change dynamically as required.

4.6.2 Data Carousel Groups and SuperGroups

A logically consistent set of Modules within the Data Carousel may be clustered together to form a Group as defined in EN 301 192 [1]. The description of the Modules in the Group is provided by a DownloadInfoIndication message. There are no restrictions on how Modules are associated into Groups and, in particular, one Module may be a member of more than one Group.

Groups may be clustered together to form a SuperGroup as defined in EN 301 192 [1]. The description of the Groups in the SuperGroup is provided by a DownloadServerInitiate message.

NOTE: A SuperGroup may contain any number of Groups, even only one.

The structure of the Data Carousel (in Groups and SuperGroups) does not necessarily reflect the structure of the content.

For purpose of clarification the exact DSM-CC messages are depicted in Annex A. Annex B gives information about the inclusion of DSM-CC messages in MPEG2 sections.

4.6.3 Use of the One-layer Data Carousel

If the Data Carousel consists only of a single Group and the complete description of the Group can be contained within a single DownloadInfoIndication message (i.e. one-layer of control information) then a one-layer Data Carousel can be used. In all other cases a two-layer Data Carousel should be used.

The DownloadInfoIndication message is the only download control message in the Data Carousel and is referred to as the top-level control message.

NOTE: Although there is only one defined download control message there may be multiple insertions of the same message in a single loop of the Data Carousel.

An example where a one-layer Data Carousel would be appropriate would be the delivery of a small HTML based application (say 10 to 20 Modules) authored to support HTML V1.0 only.

4.6.4 Use of the Two-layer Data Carousel

A two-layer Data Carousel has one or more DownloadInfoIndication messages and a single DownloadServerInitiate message (i.e. two-layers of control information). The DownloadServerInitiate message is referred to as the top-level control message.

A two-layer Data Carousel should be used in the situations described below. These are the Guidelines for specific circumstances and can be mixed together as necessary.

4.6.4.1 The Data Carousel consists of a single Group the description of which is too large for a single DownloadInfoIndication message

In this situation as many DownloadInfoIndication messages as necessary should be used to describe all the Modules in the large Group. This effectively divides the large Group up into a number of smaller Groups each defined by its own DownloadInfoIndication message. Since a Data Carousel can only have a single top-level control message this imposes the use of a two-layer carousel. To be able to recreate the original large Group the new smaller Groups need to be linked together. This is achieved by including group_link_descriptor() in the description of each of the new small Groups in the DownloadServerInitiate message.

An example would be the delivery of a large HTML based application (say 500 + Modules) authored to support HTML V1.0 only.

4.6.4.2 The Data Carousel delivers a single version of an application but supports a number of different receiver profiles

In this situation the Data Carousel should consist of a Group for each different receiver profile that is to be supported, with common Modules shared amongst more than one Group.

An example would be the delivery of a small HTML based application (say 10 - 20 Modules) authored to support HTML V1.0, V2.0 and V3.0. The Data Carousel would be structured as a SuperGroup containing three Groups. Many of the Modules would be common to all three Groups, for example GIFs and JPEGs, but some would be specific to only one Group, for example a particular HTML version of a page.

The **groupCompatability** structure associated with each Group would be used to determine which Group should be used for a given receiver profile.

4.6.4.3 The Data Carousel simultaneously delivers more than one version of an application for a single receiver profile

In this situation the Data Carousel should consist of a Group for each version of the application being delivered. Since there is no Group versioning mechanism available, the DownloadServerInitiate message should only reference the Group that describes the most recent version. This means that new viewers who access the Data Carousel via the top-level control message will automatically use this version.

If a new version of the application is to be added to the Data Carousel whilst still delivering existing versions then a new Group should be created. The DownloadServerInitiate message should be updated to remove any reference to the previous "most recent" Group and now reference the new "most recent" Group. This imposes the restriction that the receiver has to store locally the relevant top-level (DownloadServerInitiate) control message if it wishes to continue to access an older version still being broadcast.

NOTE: The **transactionId** field in the data_broadcast_descriptor could be used to point directly at the DownloadInfoIndication message that describes an older version of the Group still in the Data Carousel.

An example would be using the receiver as a software download interface to a mass storage device where it is desirable to continue to broadcast a large file to completion even though a more recent version of the same file is also being broadcast.

4.6.5 Assignment and use of transactionId values

The use of the **transactionId** in the DVB Data 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 download 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 DVB Data Carousel this should be interpreted as "changed".

The transactionId has been split up into a number of sub-fields defined in Table 4.1. This reflects the due role of the transactionId (outlined above) and constraints imposed by DVB to reduce the minimum level of filtering required by receivers. However, to increase interoperability the assignment of the transactionId has been designed to be independent of the expected filtering in target receivers.

Table 4.1: 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 top-level control message. All non-top-level control messages shall have one or more non-zero bit(s).
16 to 29	User-defined	Version	This shall be incremented/changed every time the control message is updated.
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 any message in the Data Carousel will cause the **transactionId** of the top-level control message to be incremented. The change propagates up through the structure of the Data Carousel as follows. 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 any Group(s) that includes it. Since a field in the DownloadInfoIndication message is changed its **transactionId** shall be incremented to indicate a new version of the message. Again (in the case of a two-layer Data Carousel) this change shall be reflected in the corresponding field in the description of the Group in the DownloadServerInitiate message that describes the SuperGroup. Since fields in the DownloadServerInitiate message have changed its **transactionId** shall also be incremented. This is useful since just by looking at the **transactionId** of the top-level control message a change to any message in the Data Carousel can be detected.

If the **transactionId** of any control message is referenced in the corresponding field of a data_broadcast_descriptor in SI (see EN 300 468 [6], Subclause 6.2.6) then this will need to be updated to reflect any changes. One consequence of this is that changes to the content of the Data Carousel may necessitate re-aquisition of the modified SI tables. Due to the repetition rate of SI which can be up to 2 seconds, this may be an undesired side-effect that reduces the speed of response of dynamic data services. To avoid this behaviour the value of 0xFFFFFFFF for the contents of the **transactionId** field in the data_broadcast_descriptor can be used to indicate any top-level control message is valid.

The encapsulation of download control messages within MPEG2 Transport Streams is defined in the DSM-CC specification. It specifies that the 2 least significant bytes of the **transactionId** field are copied into the **table_id_extension** field of the DSMCC_section header. This means that if the PID on which the DVB Data Carousel is being broadcast is known the top-level control message can be located without knowing its **transactionId** by setting up the section filters for **table_id** = 0x3B (download control messages) and **table_id_extension** = 0x0000 or 0x0001.

Table 4.1a reflects the encoding of the section header fields for the different message type.

Table 4.1a: Encoding of DSM-CC section_fields

Message	table_ id	table_id_extension	version_ number	section_ number	last_section_ number
Download-ServerInitiate (DSI)	0x3B	two LSB bytes of transaction_id of DSI	0x00	0x00	0x00
Download-InfoIndication (DII)	0x3B	two LSB bytes of transaction_id of DII	0x00	0x00	0x00
Download-DataBlock (DDB)	0x3C	moduleId	module Version % 32	blockNumber % 256	Max(section_ number)

4.6.6 Use of Descriptors Specific to the DVB Data Carousel

All descriptors described in this subclause are optional.

4.6.6.1 Type descriptor

With this descriptor the type of the Module or Group of the Data Carousel is transmitted. Its use is proprietary to the service provider. A string of 'char' fields specifies the type of the module following the Media Type specifications RFC 1521 [10] and RFC 1590 [11].

4.6.6.2 Name descriptor

With this descriptor the name of the Module or Group in the Data Carousel is transmitted. Its use is proprietary to the service provider.

4.6.6.3 Info descriptor

With this descriptor information about the Module or Group in the Data Carousel is transmitted. Its use is proprietary to the service provider.

4.6.6.4 Module link descriptor

The module_link_descriptor provides information about which Modules of one group are to be linked to get a complete piece of data out of the carousel. Within this descriptor two fields, the **position** field and the **module_id** field together indicate what is the first module in the list (**position** = 0x00, **module_id** = next module number), what is the next module (**position** = 0x01, **module_id** = next module number) and what is the last module (**position** = 0x02) in the list in case of a multi-module linkage.

4.6.6.5 CRC-32 descriptor

With this descriptor CRC-32 calculation over a complete Module is indicated. The CRC-32 bits of the Module are part of the descriptor.

4.6.6.6 Location descriptor

The location descriptor in a DownloadServerInitiate message indicates on which PID a Group of the Data Carousel can be found. The DownloadInfoIndication message of the Group to be found has to be on that PID. The same mechanism can be used in the DownloadInfoIndication message to find all the Modules on different PIDs.

This is a very powerful means to operate with Groups and Modules for different kinds of users.

4.6.6.7 Estimated download time descriptor

The descriptor for estimated download time has been introduced in order to provide an indication to the receiver of the time it will take to download a Module or Group.

Some care is needed in how it is used. The download time will obviously be sensitive to the bitrate available to deliver the Data Carousel. This may be a problem where the Data Carousel is produced separately from playout of that carousel. If playout of the same Data Carousel is at one bitrate on one day (for example 1 Mbit/s) and at another bitrate on the next day (for example 2 Mbit/s) then the estimated download time can not be correct for both (or even either!).

NOTE: One approach would be to calculate the value for estimated download time based on the minimum playout bitrate. Obviously it may be more practical in some cases for the receiver to simply indicate how much of the data has been received based on received messages.

4.6.6.8 Group link descriptor

The description of the Modules in a Group is provided by a DownloadInfoIndication message. The number of Modules that may be described is determined by the maximum size of such a message and the size of the description of each Module. The encapsulation of such download control messages within MPEG2 sections limits the maximum size to just under 4 Kbytes. The size of a simple Module description (say basic information and a name descriptor of 30 bytes) is about 40 bytes. This means that the DownloadInfoIndication message can describe about 100 Modules which will be sufficient in most cases but not all.

In the later situation as many DownloadInfoIndication messages as necessary should be used to describe all the Modules in the large Group. This effectively divides the large Group up into a number of smaller Groups each defined by its own DownloadInfoIndication message. To be able to recreate the original large Group the new smaller Groups need to be linked together. This is achieved by including group_link_descriptor() in the description of each of the new small Groups in the DownloadServerInitiate message.

4.6.6.9 Private descriptor

If a service provider has a need for a private descriptor the syntax of the private descriptor in (EN 301 192 [1], Subclause 7.2.10) shall be used.

4.6.6.10 Compressed module descriptor

Presence of the compressed_module_descriptor indicates that the data in the module has the "zlib" structure as defined in RFC 1951 [14]. The ZLIB structure is defined as:

Table 4.1b

zlib structure(){	Number of bytes
compression_method	1
flags_check	1
compressed_data	n
check value	4
}	

4.6.7 Information in the SI and PSI

Access to the Data Carousel can be achieved via descriptors in either SI or PSI. This provides some flexibility in how the service is identified.

4.6.7.1 Descriptor in SI

For services using Data Carousel(s), the data_broadcast_descriptor shall be present in the SDT or the EIT, i.e. use of this descriptor is mandatory.

The **data_broadcast_id** field shall be set to 0x0006 to indicate the use of the DVB Data Carousel.

The **component_tag** will identify the PID on which the Data Carousel is broadcast by association with a similar tag in the stream_identifier_descriptor() for the particular stream in the PMT.

The **data_carousel_info** structure EN 301 192 [1] is carried in the selector_byte field.

The **carousel_type_id** indicates which kind of Data Carousel is used (Figure 7.1 EN 301 192 [1]).

The use of the **transaction_id** is depicted in Subclause 4.6.4.

The **time_out_value_DSI** and **time_out_value_DII** gives some indication to the receiver of how long it shall wait before assuming an error condition.

The **leak_rate** is included for optimization of the receiving device. By giving the **leak_rate** a decoder is able to compute whether a service can be decoded. The leak rate may also be given in a smoothing_buffer_descriptor or a maximum_bitrate_descriptor in which case the values given in both descriptors shall be consistent. However, the usage of a maximum bitrate descriptor is not recommended.

The advantages of using an SI based access to the carousel instead of the PSI one are:

- The **transactionId** can be used to explicitly identify the top-level control message in the Data Carousel.
- By including the **transactionId** field in this descriptor, updates to the Data Carousel (which will cause a change in **transactionId**) can be detected by filtering on just the SI.

  NOTE: This behaviour can be avoided by using the special value of **transactionId**, 0xFFFFFFFF, as described in Subclause 4.6.4.

- The descriptor does not consume any space in the PSI tables (which may be a scarce resource).

The disadvantage of using an SI based access to the carousel instead of the PSI one is:

- The repetition period of SI can be up to 2 seconds which can introduce delay to the initial access of the service.

4.6.7.2 Descriptors in PSI

For services using Data Carousel(s), the data_broadcast_id_descriptor can be present in the PMT, i.e. use of this descriptor is optional.

The **data_broadcast_id** field shall be set to 0x0006 to indicate the use of the DVB Data Carousel.

The advantage of using this mechanism is that:

- The maximum repetition period of PSI is only 0,1 seconds which allows fast initial access to the service.

The disadvantages of this mechanism are that:

- There is no **transactionId** field so explicitly identify the top-level control message. As such only download control messages from a single Data Carousel may be transported on the identified elementary stream.
- The descriptor does not provide any information about the time-out period for download control messages. This information shall still be obtained from the descriptor in SI.
- The descriptor consumes some space (albeit small) in the PSI tables.
- The descriptor in SI shall still be included as well.

4.7 Object Carousel

4.7.1 Introduction

A DSM-CC Object Carousel facilitates the transmission of a structured group of objects from a broadcast Server to broadcast Receivers (Clients) using directory objects, file objects and stream objects. The actual directory and content (object implementations) are located at the Server. The Server repeatedly inserts the mentioned objects in the DVB compliant MPEG2 Transport Stream using the Object Carousel protocol. The Object Carousel is part of a DVB Service as shown in Figure 4.4. The transmitted directory and file objects contain the content of the objects, while the transmitted stream objects are references to other streams in the broadcast. The stream objects may also contain information about the DSM-CC events that are broadcast within a particular stream. DSM-CC events can be broadcast with regular stream data and can be used to trigger DSM-CC applications.

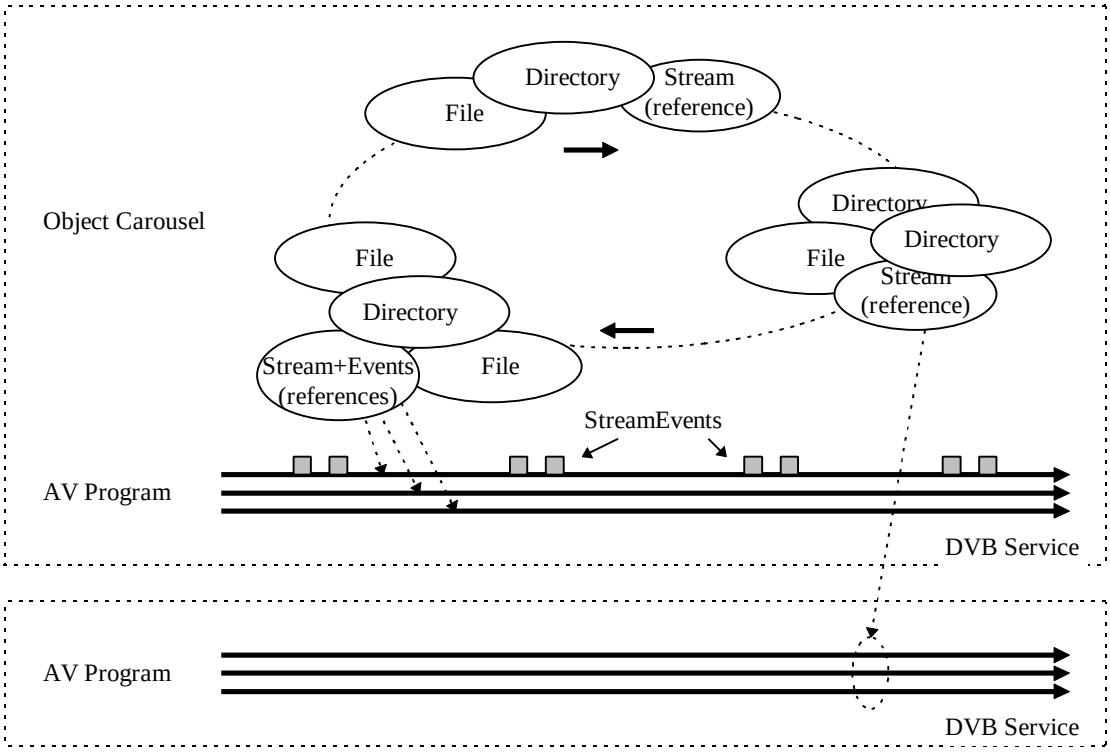


Figure 4.4: Example of including Object Carousel specification in DVB Services

Multiple Clients can recover the object implementations by reading the repeatedly transmitted carousel data, hence mimicking the Server's objects in a local object implementation. The Objects in the carousel offer Clients a way to access applications and content used by these applications, more or less as if there was an interactive connection with the Server.

The following sections provide guidelines regarding the implementation and use of DSM-CC U-U Object Carousels in DVB-compliant broadcast networks and in interactive systems compliant to ETS 300 802 [2]. This Clause focuses on the following subjects:

- Platform independence;
- Encoding of BIOP control structures used in U-U Object Carousels;
- Encoding of BIOP data messages used in U-U Object Carousels;
- Encoding of Download Data Carousel messages;
- Encoding of DSM-CC sections;
- Use of PSI descriptors for Object Carousels; and
- Use of SI descriptors for Object Carousels.

The scope is illustrated in Figure 4.5 by the area surrounded by thick lines. Figure 4.5 shows the protocol stacks defined by DVB-SIS for both Broadcast and Interactive Networks.

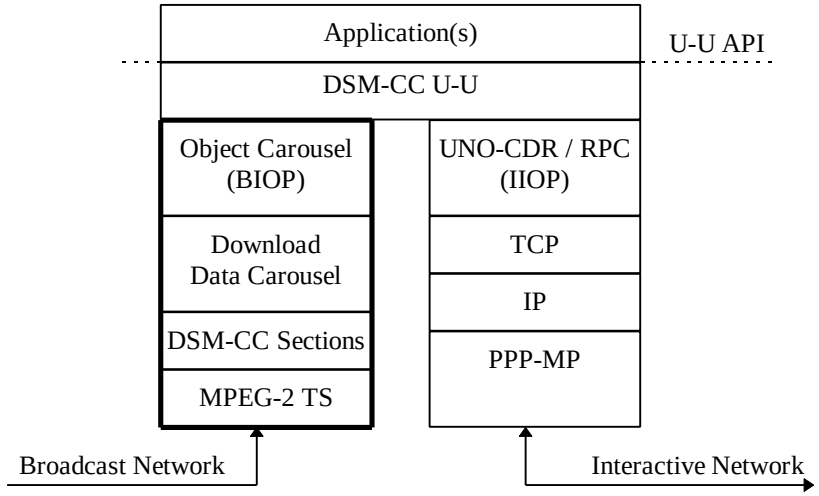


Figure 4.5: Place of object carousel protocols in the DVB-SIS framework

4.7.2 Platform independence

4.7.2.1 Overview

The Object Carousel specification is platform-independent and compatible with the DSM-CC User-to-User specification of ISO/IEC 13818-6 [4] and with the Object Request Broker (ORB) framework as defined by CORBA [9]. Within the DSM-CC User-to-User (U-U) system environment, a structured group of objects is referred to as a Service Domain. The Service Domain has a Service Gateway which can be seen as the top-level directory of the structured group of objects. As such the Service Gateway provides a context for the graph of service names (i.e. object names) that is published to the Clients. A Service Domain can be located at a Server in an interactive network as well as on a Server in a broadcast Network. In the latter case the objects within the Service Domain are broadcast by means of an Object Carousel.

NOTE: For naming of objects please refer to Annex C of the present document.

The data and attributes of a single Object within an Object Carousel are transmitted in a single message. The message format is specified by the Object Carousel specification and is referred to as the BIOP message format (Broadcast Inter ORB Protocol). BIOP messages are broadcast in a single Module of a DSM-CC Data Carousel (ISO/IEC 13818-6 [4]). One Module may contain one or more BIOP messages. According to the DSM-CC Data Carousel specification each Module is fragmented into one or more Blocks which are carried in a DownloadDataBlock message. Each DownloadDataBlock message is of the same size (except for the last block of the Module which may be of a smaller size) and is transmitted in turn in an MPEG2 section as specified in ISO/IEC 13818-6 [4]. The encapsulation rules for DownloadDataBlock messages in MPEG2 sections are such that Blocks can be acquired directly from the Transport Stream using hardware filters found generally on demultiplexers.

Objects within Service Domains are identified using object references. DSM-CC U-U uses the Interoperable Object Reference (IOR) structure as defined by CORBA. The object reference contains all the information that is necessary to retrieve the object from one or more Servers in the network. The structure in the IOR that contains the location information of a single instance of a stored Object is called a profile body. An IOR may contain multiple Profile Bodies to indicate multiple (network) locations of the object.

The Object Carousel specification uses two Profile Bodies. These two Profile Bodies: BIOPProfileBody and LiteOptionsProfileBody, are used to refer to objects that are located either in the same Object Carousel or in another Object Carousel, respectively.

The first Profile Body is called the Broadcast Inter ORB Protocol (BIOP) Profile Body and is solely used to refer to objects within the same Object Carousel (i.e. Service Domain). It facilitates the unique identification of the Object using the identifier of the Object Carousel, the identifier of the Module in which the object is broadcast, and an unique key that identifies the object within the Module. The identifier of the Object Carousel is linked to a DVB-service via a descriptor in the PMT of the MPEG program.

The second Profile Body is called the Lite Options Profile Body and is used to refer to objects in another Service Domain (either Interactive or Broadcast). It facilitates applications to connect to another Service Domain using a globally unique NSAP address format. For Service Domains in DVB-compliant networks the NSAP address is linked to a particular DVB-service.

4.7.2.2 Supported U-U Objects

The Object Carousel specification is designed to support a number of the interfaces defined in the Application Portability Interface (API) of DSM-CC U-U (User-to-User). This subclause provides guidelines regarding the objects and interfaces supported within Object Carousels (see for interface definitions ISO/IEC 13818-6 [4]):

Table 4.2: Objects with supported READER interfaces

Object	Supported READER Interfaces
DSM::Directory	Access, Directory
DSM::File	Base, Access, File
DSM::Stream	Base, Access, Stream
DSM::ServiceGateway	Access, ServiceGateway
BIOP::StreamEvent	Base, Access, Stream, Event

It should be noted that the semantics of the API for broadcast networks will differ slightly from the semantics of the API for interactive networks. The cause for this lies in the broadcast nature of the network. A typical example is with the Stream interface where a pause ("now") API call for streams delivered via the broadcast network may freeze the image on screen but not pause the delivery of the (broadcast) stream.

DVB Guideline: *The present document does not provide any guidelines regarding the precise operation of the DSM-CC U-U interface in Broadcast networks.*

The DSM-CC interface Access will return attributes (i.e. object properties like read permission and access times) which are set to default values because the broadcast of these attributes is not defined in BIOP (ISO/IEC 13818-6 [4], ISO/IEC 13818-1 [3]).

DVB Guideline: *The present document does not provide any guidelines regarding the broadcasting of Access attributes in Object Carousel.*

Figure 4.6 shows the relationships between the U-U Objects using OMT notation [12].

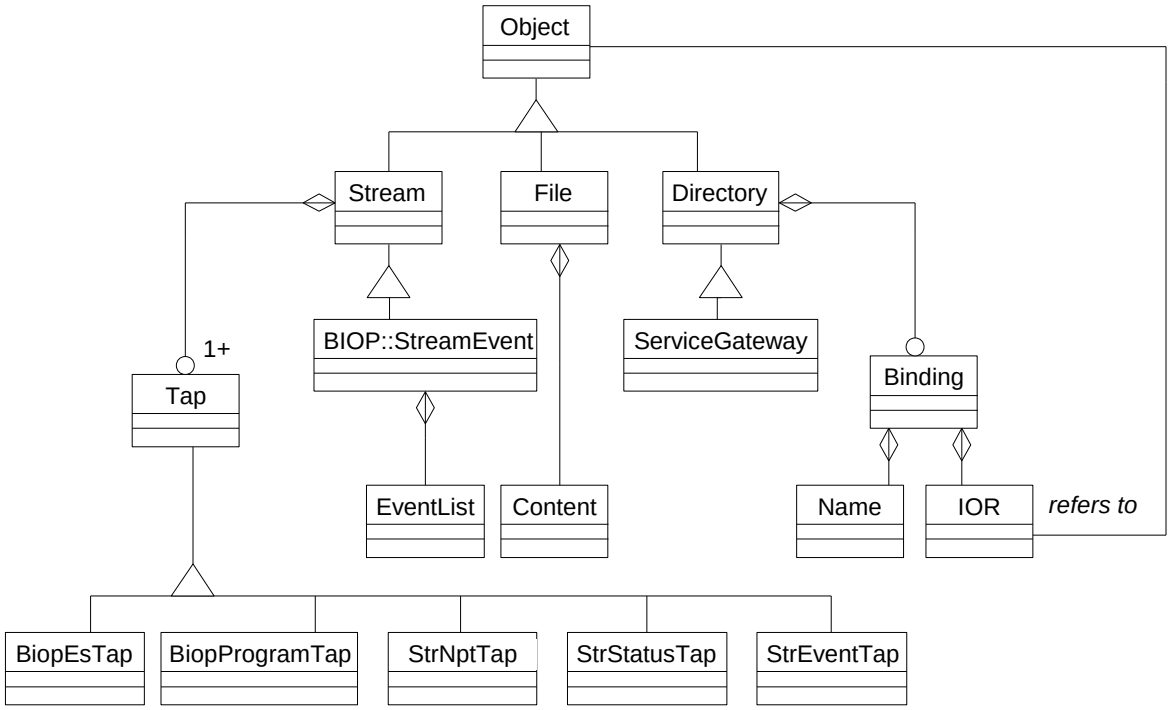


Figure 4.6: Supported Objects within Object Carousel

In an Object Carousel the following information is transmitted for each object:

Directory object data:	List of *Bindings*, where each Binding binds a Name to an object reference (IOR). In addition, each Binding may also contain some additional attributes of the bound object to support the fast browsing through directories. In the current Object Carousel specifications this is only used for the contentSize attribute for file objects.
File object data:	File content data and the contentSize attribute.
Stream object data:	A list of identifiers (called Taps) referring to one or more streams in the Broadcast network. Each Tap refers to either an Elementary Stream (BiopEsTap) or to a complete MPEG program (BiopProgramTap). Additionally other identifiers may be present that point to broadcast channels that contain control information for the stream (such as Taps that refer to StreamDescriptors for NPT, status/mode and events). The stream object data also includes the StreamInfo attribute.
ServiceGateway object data:	Identical to Directory object because ServiceGateway inherits from Directory. Special for the ServiceGateway object is that it contains the Root directory of the Service Domain.
StreamEvent object data:	Similar to the Stream object data, but extended with the EventList attribute and a list of eventIds. These attributes contain a list of DSM-CC event names and a mapping of those to eventIds.

4.7.2.3 Transmission of objects

The data and attributes of one U-U Object in an Object Carousel are transmitted in one message. The message format is specified by the Broadcast Inter ORB Protocol (BIOP) and is referred to as the BIOP Generic Object Message format (or BIOP message for short). A BIOP Message consists of a MessageHeader, a MessageSubHeader and a messageBody. The MessageHeader provides information about the version of the BIOP protocol and the length of the BIOP message. The MessageSubHeader contains information about the conveyed Object such as objectType (File,

Stream, Directory) and objectKey (the unique identifier within a Module). The messageBody depends on the objectType and contains the actual U-U Object's data. The size of a BIOP message is variable.

BIOP messages are broadcast in Modules of Data Carousels (ISO/IEC 13818-6 [4]). A Module is formed by the one or more concatenated BIOP Messages (see Figure 4.7) and are thus of variable length. Within the Module each Object is identified by the objectKey. An Object can easily be found by parsing subsequently the objectKey field of the BIOP message and the length of the BIOP message.

According to the DSM-CC Data Carousel specification each module is fragmented into one or more Blocks which are carried in a DownloadDataBlock message. Each DownloadDataBlock message is of the same size (except for the last block of the Module which may be of a smaller size) and is transmitted in turn in an MPEG2 private section as specified in ISO/IEC 13818-6 [4]. The encapsulation rules for DownloadDataBlock messages in MPEG2 private sections are such that Blocks can be acquired directly from the Transport Stream using hardware filters found generally on demultiplexers.

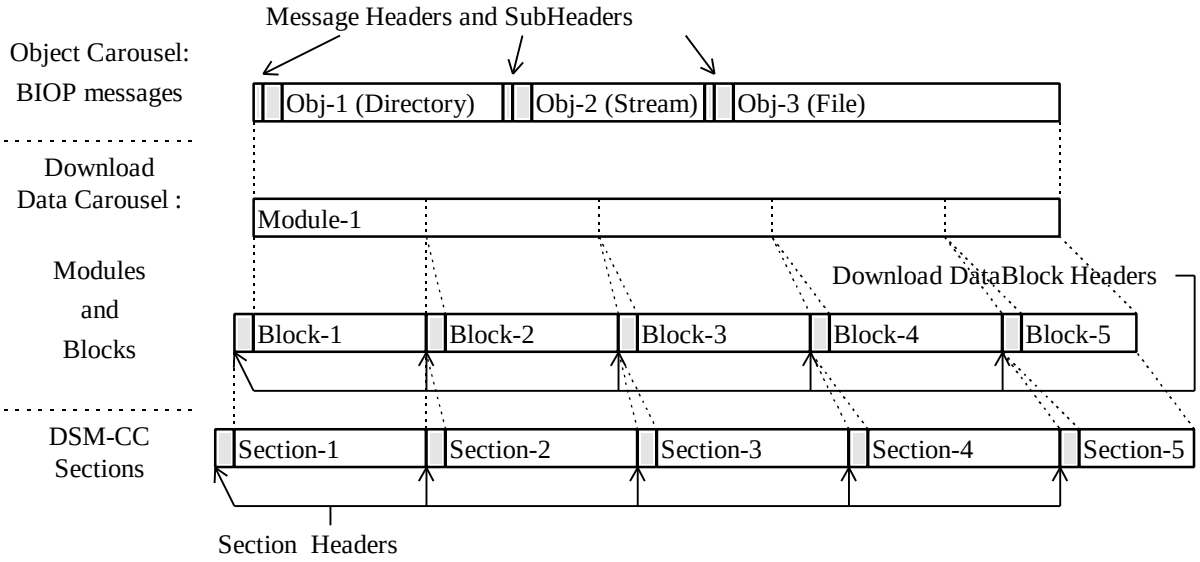


Figure 4.7: Encapsulation and fragmentation of BIOP Messages in Modules, Blocks, and MPEG2 sections

The acquisition of an object from the broadcast network requires the complete acquisition of the module in which the object is contained. This requires knowledge of the delivery parameters of the Module such as module version, module size, block size, timing and broadcast channel. These delivery parameters are transmitted in a DownloadInfoIndication message which has to be acquired from the network before acquiring the module ISO/IEC 13818-6 [4]. One DownloadInfoIndication message can describe the delivery parameters of multiple modules. The retrieval of an object from the Broadcast network is therefore a two-step process.

Within BIOP the object reference of the Service Gateway of a Service Domain is transmitted in a DownloadServerInitiate message (ISO/IEC 13818-6 [4]). This message can be found using information from either the PSI or the PSI **and** SI.

4.7.2.4 Object References

BIOP uses CORBA's Interoperable Object Reference (see also ISO/IEC 13818-6 [4] and [9]). An object reference contains for each network location one Profile Body. The type of Profile Body depends on the protocols that are necessary to acquire the Object from the Server.

For an IOR that refers to an Object within the same broadcast Service Domain (i.e. within the same Object Carousel), the BIOP Profile Body identifies the location of the BIOP message that conveys the Object data and attributes. The BIOP Profile Body consists therefore of an ObjectLocation component and a ConnBinder component (See Figure 4.8).

Figure 4.8 illustrates how the object reference (IOR) with BIOP Profile Body can be resolved into the Object that it refers to. The ObjectLocation identifies the object in the U-U Object Carousel by means of the triple *carouselId*,

moduleId and *objectKey*. The ConnBinder consists of a sequence of *Taps*. The Taps identify via the PMT the streams on which the DownloadInfoIndication message is broadcast that contains the Module Delivery Parameters of the object.

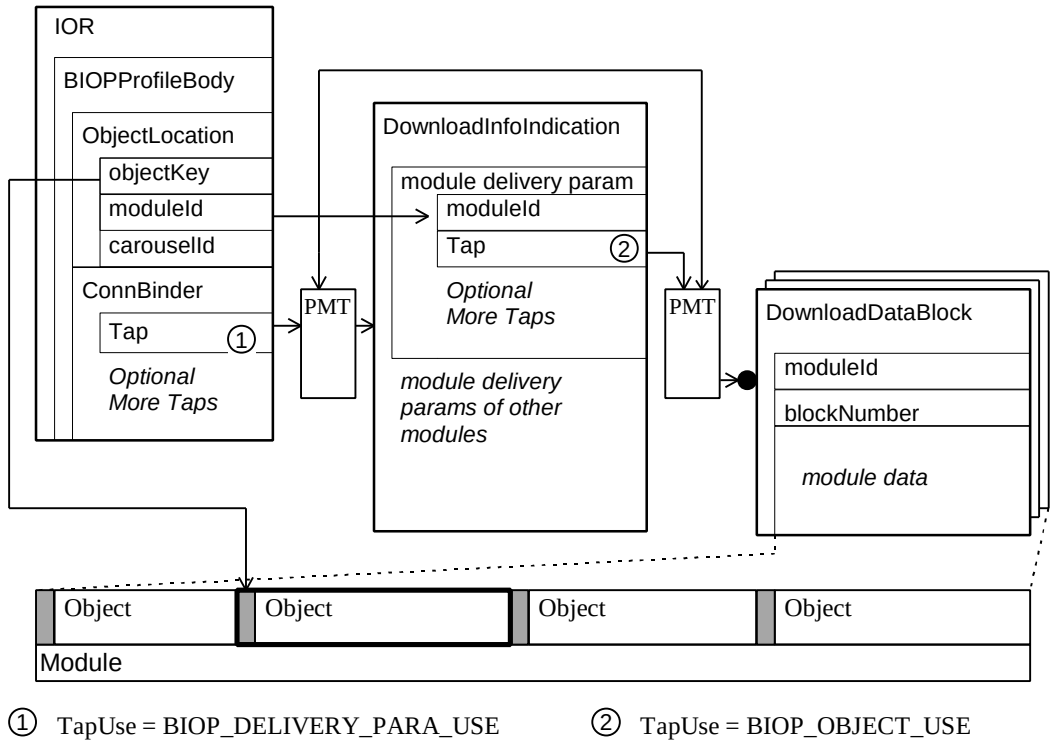


Figure 4.8: How an IOR with BIOP profile body can be resolved into an Object

The ConnBinder shall contain at least one Tap that "points" via the PMT to the DownloadInfoIndication message. The moduleId in the IOR is used to determine the appropriate delivery parameters in the DownloadInfoIndication message. The delivery parameters shall in turn contain at least one Tap that "points" (also via the PMT) to the DownloadDataBlock messages that convey the Module. Finally the objectKey from the IOR is used to identify the Object message in the Module.

NOTE: Both the ConnBinder and the module delivery parameters may contain more than one Tap. Additional Taps may identify alternative streams where the same Module (with possible other delivery parameters) is transmitted.

For an IOR that refers to an object in another Service Domain the Lite Options Profile Body is used. The Lite Options Profile Body uses a globally unique NSAP address to identify the Service Domain which may be either Interactive or Broadcast. For Service Domains in DVB-compliant broadcast networks the NSAP address identifies a particular DVB-service as specified in EN 301 192 [1] (See Figure 4.9).

Figure 4.9 illustrates how the object reference (IOR) with a Lite Options Profile Body can be resolved into the Service Gateway of a broadcast Service Domain. The Profile Body contains a Service Location component that contains in turn the NSAP address. The NSAP address identifies the broadcast Service Domain using the triple *transport_stream_id*, *service_id*, and *orginal_network_id* of the DVB service in which the Object Carousel is broadcast. Using the PAT and the PMT of the service the IOR of the Service Gateway is found in a DownloadServerInitiate message. This IOR contains in turn an BIOP Profile Body that points to the Service Gateway Object of the broadcast Service Domain. The resolve operation of the BIOP Profile Body is identical as in Figure 4.8.

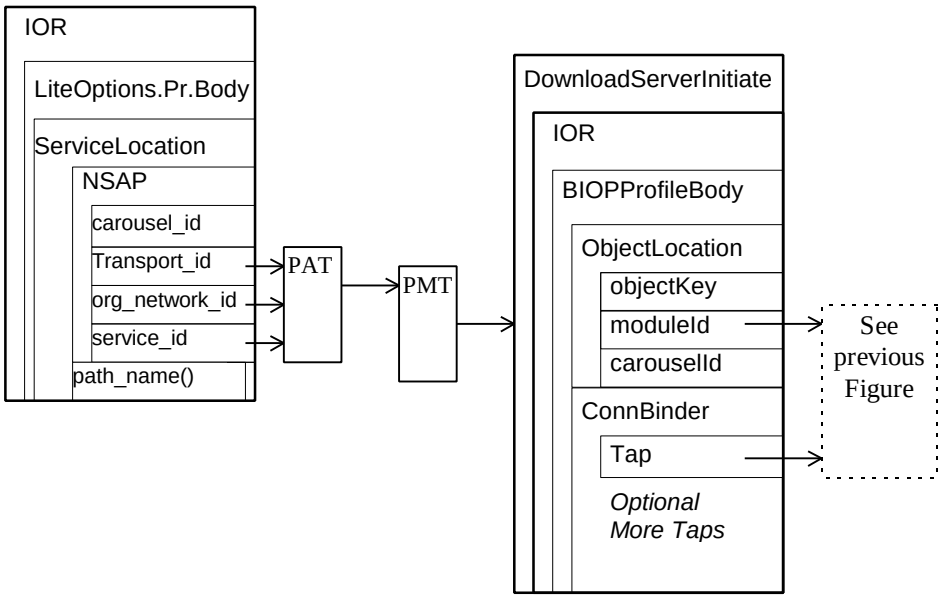


Figure 4.9: How an IOR with Lite Options Profile Body can be resolved into a Service Gateway

4.7.2.5 Taps and associations

IORs do not refer to streams directly by means of a PID, because PIDs can be changed by re-multiplexers. DSM-CC has defined therefore Taps (ISO/IEC 13818-6 [4]) which are used in a similar way as component tags in DVB SI (EN 300 468 [6]).

A Tap consists of:

- id: this field is for private use (shall be set to zero if not used);
- use: field indicating the usage of the Tap;
- association_tag: (association tag) field to associate the Tap with a particular (Elementary) Stream;
- selector: optional selector, to select the associated data from the associated (Elementary) Stream. The presence of the selector depends on the use field.

The following *use* values are used within Object Carousels:

1. BIOP_DELIVERY_PARA_USE: The ConnBinder component of an BIOP Profile Body shall include such Taps to indicate the connections at which the DownloadInfoIndication() messages are broadcast that describe the module delivery parameters of the Module in which the object is conveyed (See Figure 4.10). The selector field of such Taps contains a transactionId field and a timeout field. The value of the transactionId field shall be set to the transactionId of the DownloadInfoIndication() message that contains the module delivery parameters. The timeout field shall be set to the time-out period in microseconds to be used to time out the acquisition of the DownloadInfoIndication message.

2. BIOP_OBJECT_USE: Used in the DownloadInfoIndication() messages which convey the module delivery parameters of the Modules to indicate the elementary stream on which the Modules are broadcast. The selector field is empty.

3. BIOP_ES_USE, BIOP_PROGRAM_USE: The Stream object contains such Taps to indicate the stream(s) that are associated with the object. Where a BIOP_ES_USE refers to a single Elementary Stream and BIOP_PROGRAM_USE refers to a complete MPEG2 Program (DVB Service). The selector field of both Tap types is empty.

4. STR_STATUS_AND_EVENT_USE, STR_EVENT_USE, STR_STATUS_USE, STR_NPT_USE: The Stream object and StreamEvent object may contain these Taps to indicate the various sub-streams that are associated with the object. The selector field of all such Taps is empty.

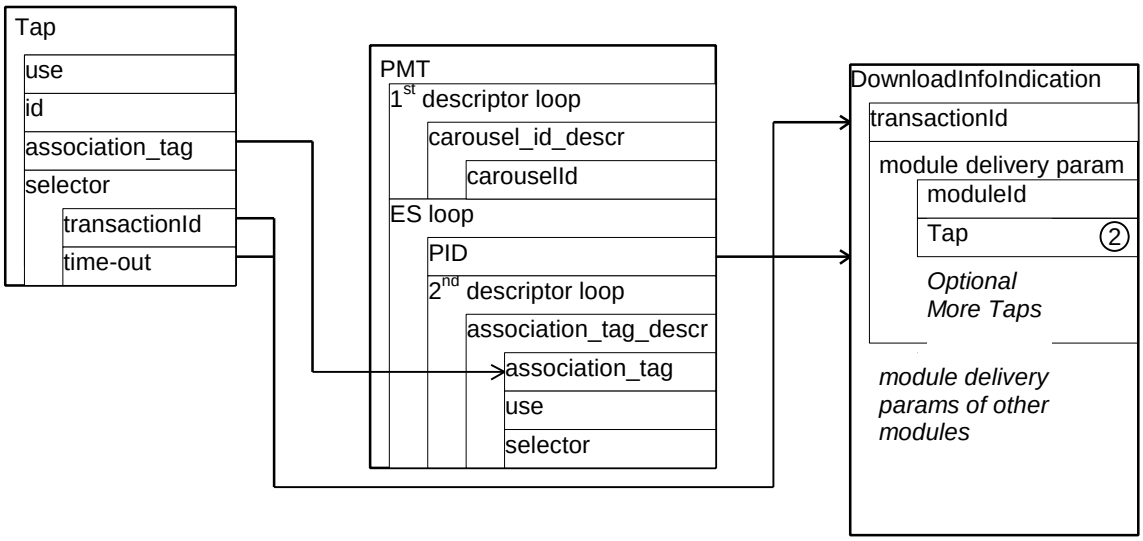


Figure 4.10: Use of association_tag descriptor to indicate elementary streams (TapUse = BIOP_DELIVERY_PARA_USE)

In the course of resolving an object, Clients have to associate the Taps to the connections of the broadcast network. Clients need, therefore, an association table that makes the association between the Taps and the connections of the broadcast network. To support the implementation of U-U Object Carousels in Broadcast Networks based on MPEG2 Transport Streams, ISO/IEC 13818-6 [4] defines three descriptors that can be inserted into MPEG2 PMTs:

1. The *carousel_identifier_descriptor* labels a PMT with a carousel_id, identifying that all association_tags present in the PMT belong to that U-U Object Carousel (providing a scope for the association tags (See Figure 4.10).

2. The *association_tag_descriptor* labels an Elementary Stream with an association_tag, associating all Taps containing this tag with this Elementary Stream (See Figure 4.10). Like a Tap, an association_tag_descriptor also contains a use field and an optional selector field. Setting this use field to 0x0000, labels the Elementary Stream that a DownloadServerInitiate message (DSI) is transmitted at this stream. This DSI contains the IOR of the ServiceGateway.

3. The *deferred_association_tags_descriptor* contains a list of association_tags that are associated with (Elementary Streams in) another MPEG2 program (PMT) or that refer to another program (for use with BIOP_PROGRAM_USE Taps). Figure 4.11 illustrates the use of the deferred_association_tags_descriptor to point to another program.

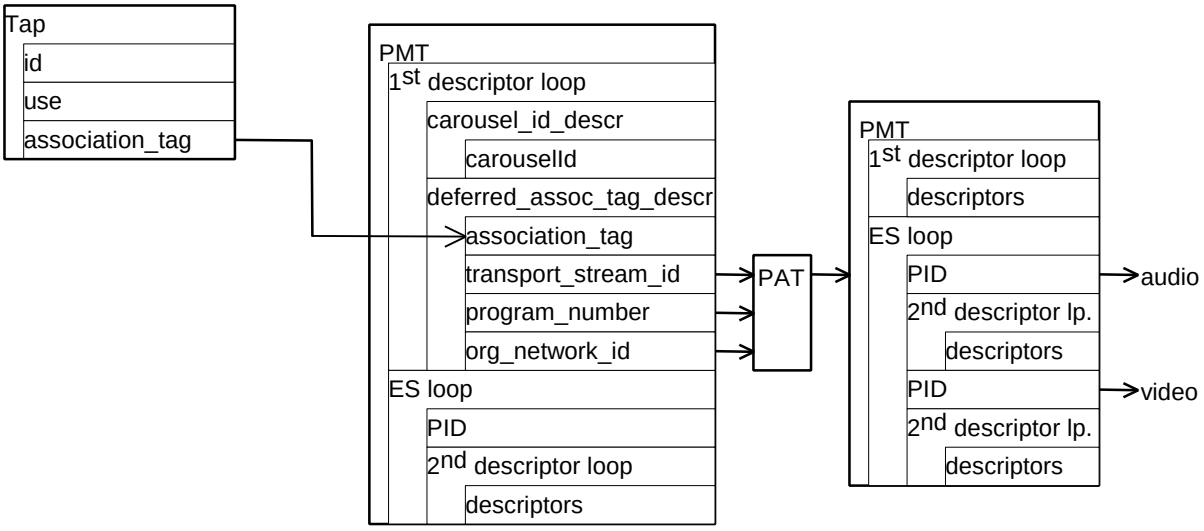


Figure 4.11: Use of deferred_association_tag descriptor to indicate an MPEG2 program (TapUse = PROGRAM_USE)

4.7.3 BIOP Control Structures

BIOP control and data structures are defined in ISO/IEC 13818-6 [4] using the platform-independent specification language OMG IDL (Interface Definition Language) as defined in [9]. The "bits-on-the-wire" encoding is defined by the CDR (Common Data Representation, [9]) encoding rules that converts IDL grammar to bits on the wire. BIOP uses the CDR Lite encoding rules (ISO/IEC 13818-6 [4]) which uses maximum length numbers in sequences and byte alignment. Consequently, CDR Lite achieve a much more compact packing of data, compared to CDR.

NOTE: That this also implies that all strings are terminated by a null character and that this character forms part of the string length (For an example see in Table 4.9 the fields objectKind_length and objectKind_data).

In this section the BIOP control structures are shown using an MPEG2 syntax and guidelines are provided concerning the encoding of the fields. Fields that are affected by the guidelines are shaded. In Section 0 the BIOP messages are shown using an MPEG2 syntax. In the case of any differences between the IDL structures defined in ISO/IEC 13818-6 [4] and the structures defined in the following sections, the defined structures in ISO/IEC 13818-6 [4] will be correct.

4.7.3.1 Interoperable Object Reference (IOR)

DSM-CC uses the Interoperable Object Reference (IOR) format defined by OMG for object references at the Client-Server Inter-operability Interface. Table 4.3 shows the syntax of the IOP::IOR (See ISO/IEC 13818-6 [4]).

Table 4.3: IOP::IOR syntax

Syntax	bits	Type	Value	Comment
IOP::IOR {				
type_id_length	32	uimsbf	N1	
for (i = 0; i < N1; i++) {				
type_id_byte	8	uimsbf	+	see Table 4.4
}				
if (N1 % 4 ≠ 0) {			+	CDR alignment rule
for (i= 0; i < (4 - (N1 % 4)); i++) {				
alignment_gap	8	uimsbf	0xFF	
}				
}				
taggedProfiles_count	32	uimsbf	N2	Profile bodies
for (n = 0; n < N2; n++) {				
IOP::taggedProfile() {				
profileId_tag	32	uimsbf	+	For example TAG_BIOP For example TAG_LITE_OPTIONS
profile_data_length	32	uimsbf	N3	
for (i = 0; i < N3; i++) {				
profile_data_byte	8	uimsbf		For example BIOPProfileBody For example LiteOptionsProfileBody
}				
}				
}				
}				

The **type_id_byte** fields of the IOR form a string representing the type of the object. For object identification in [9] mechanisms, string ids are used in the form "<Module>::<Interface>". In order to reduce the size of IORs, DSM-CC defines aliases of 3 characters. The type_ids for Objects used in a DVB Object Carousels are shown in Table 4.4.

Table 4.4: U-U Objects type_id

Full type_id	alias type_id
”DSM::Directory”	”dir”
”DSM::File”	”fil”
”DSM::Stream”	”str”
”DSM::ServiceGateway”	”srg”
”BIOP::StreamEvent”	”ste”

DVB Guideline: Only the alias type_id fields shall be used with DVB compliant systems. This implies that no alignment stuffing bytes have to be inserted by the Server when using these aliases.

An IOR that refers to an object transmitted in the same U-U Object Carousel contains a BIOP Profile Body in the **taggedProfileList**. ISO/IEC 13818-6 [4] allows an IOR to contain more than one profile body.

DVB Guideline: DVB compliant receivers shall be able to process at least the first of these profile bodies, while the other profile bodies may be ignored.

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.

4.7.3.2 BIOP Profile Body

The BIOP Profile Body has a LiteComponentProfile structure which follows the MultipleComponentProfile structure. Table 4.5 shows the syntax of the BIOP Profile Body including the mandatory ObjectLocation component and ConnBinder Component.

Table 4.5: BIOP Profile Body syntax

Syntax	bits	Type	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
liteComponents_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	
for (k = 0; k < N2; k++) {				
objectKey_data_byte	8	uimsbf	+	
}				
}				
DSM::ConnBinder {				
componentId_tag	32	uimsbf	0x49534F40	TAG_ConnBinder
component_data_length	8	uimsbf	*	
taps_count	8	uimsbf	N3	
BIOP::Tap {				
id	16	uimsbf	0x0000	user private
use	16	uimsbf	0x0016	BIOP_DELIVERY_PARA_USE
association_tag	16	uimsbf	+	
selector_length	8	uimsbf	0x0A	
selector_type	16	uimsbf	0x01	
transactionId	32	uimsbf	*	
timeout	32	uimsbf	*	
}				
for (m = 0; m < N3 - 1; m++) {				
BIOP::Tap {				
id	16	uimsbf	0x0000	user private
use	16	uimsbf	0x0016	BIOP_DELIVERY_PARA_USE
association_tag	16	uimsbf	+	
selector_length	8	uimsbf	N4	
for (i = 0; i < N4; i++) {				
selector_data_byte	8	uimsbf		
}				
}				
}				
}				
for (n = 0; n < N5;n++) {				N5 = N1 - 2
BIOP::LiteComponent {				
componentId_tag	32	uimsbf	+	
component_data_length	8	uimsbf	N6	
for (i = 0; i < N6; i++) {				
component_data_byte	8	uimsbf		
}				
}				
}				
}				

DVB Guideline: The byte_order field shall have the value of 0x00 meaning that following data is encoded using big-endian byte ordering.

The **carouselId** field provides a context for the moduleId field. It uniquely identifies the carousel within the Broadcast Network and allows the distributed implementation of the carousel.

DVB Guideline: The BIOP Profile Body shall only be used to refer to Objects within the same carousel. I.e. the value of the carouselId is equal to the carouselId of the Object Carousel in which the IOR is transmitted. To refer to Objects in another carousel use the Lite Options Profile Body.

DVB Guideline: The list of LiteOptionComponents shall contain exactly 1 BiopObjectLocation and exactly 1 DsmConnectionBinder as the first two components in that order.

The **moduleId** identifies the module in which the object is conveyed within the carousel.

The **objectKey** identifies the object within the module in which it is broadcast. This field is a series of bytes that is supplied by the server and which is only meaningful to the server.

DVB Guideline: The value of the objectKey length field shall be less than or equal to 0x04.

Multiple Taps may share the same association tag, enabling one Elementary Stream to be used for more than one purpose. Table 4.6 shows the defined Tap uses.

Table 4.6: Allowed Tap use definitions for Taps in a BIOP Profile Body

TapUse field	Value	Broadcast on PID
BIOP_DELIVERY_PARA_USE	0x16	Module delivery parameters
BIOP_OBJECT_USE	0x17	BIOP objects in Modules

DVB Guideline: If the BIOP_DELIVERY_PARA_USE tap is present in the ConnBinder component then it will be the first tap in the ConnBinder.

DVB Guideline: DVB compliant receivers may skip over the BIOP_OBJECT_USE taps in BIOP Profile Bodies in IORs.

DVB Guideline: The id field shall be set to zero if not used.

The semantics of the fields of a Tap with a TapUse value of BIOP_DELIVERY_PARA_USE are described below:

The use field indicates the use of the Tap. The value of this field shall be BIOP_DELIVERY_PARA_USE.

The association_tag identifies the broadcast channel (i.e. the Elementary Stream) on which the DownloadInfoIndication() message is broadcast.

The selector field shall contain a selectorType of value MESSAGE (= 0x0001) and the transactionId and timeout fields. The value of the transactionId field shall be set to the transactionId of the DownloadInfoIndication() message that contains the module delivery parameters. The timeout field shall indicate the time-out period in microseconds to be used to time out the acquisition of the DownloadInfoIndication() message.

The semantics of the fields of a Tap with a TapUse value of BIOP_OBJECT_USE are described below:

- The use field indicates the use of the Tap. The value of this field shall be BIOP_OBJECT_USE.
- The association_tag identifies the broadcast channel (i.e. Elementary Stream) on which the Modules are broadcast.
- The selector field shall be of 0 length.

NOTE: Taps with a TapUse value of BIOP_OBJECT_USE should, however, in DVB compliant systems be used only in the DownloadInfoIndication messages and not in the IORs.

4.7.3.3 Lite Options Profile Body

To refer to an Object in another Service Domain, an IOR is present that contains a ServiceLocation component in an Lite Options Profile Body. When a DSM-CC U-U API user attempts to resolve a Name (Directory::resolve, see ISO/IEC 13818-6 [4], Clause 5]), that results in the encounter of such an IOR, a

SERVICE_XFR exception is raised. A SERVICE_XFR exception carries the ServiceLocation structure found in the Lite Options Profile Body of the IOR. The API user may use the serviceDomain from the ServiceLocation structure for a subsequent attach to the new ServiceGateway. The optional pathName contains the path within that ServiceGateway to find the Object.

A Lite Options Profile Body has a LiteComponentProfile structure which follows the MultipleComponentProfile structure. Table 4.7 shows the syntax of an Options Profile Body, that conveys a ServiceLocation component.

Table 4.7: Syntax of Options Profile Body with ServiceLocation component

Syntax	bits	Type	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
component_count	8	uimsbf	N1	
DSM::ServiceLocation {				
componentId_tag	32	uimsbf	0x49534F46	TAG_ServiceLocation
component_data_length	32	uimsbf	*	
serviceDomain_length	8	uimsbf	0x14	Length of carousel NSAP address
serviceDomain_data()	160	uimsbf	+	DVBcarouselNSAPaddress see Table 4.8
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_byte	8	uimsbf	+	
}				
kind_length	32	uimsbf	N4	NameComponent kind
for (j = 0; j < N4 j++) {				
kind_data_byte	8	uimsbf	+	as type_id (see Table 4.4)
}				
}				
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::LiteOptionComponent{				
componentId_tag	32	uimsbf	+	
component_data_length	8	uimsbf	N7	
for (i = 0; i < N7; i++) {				
component_data_byte	8	uimsbf		
}				
}				
}				
}				

DVB Guideline: The ServiceLocation component shall be the first component in the profile body.

4.7.3.4 Carousel NSAP address

Each instance of a U-U Object Carousel represents a Service Domain. Each Service Domain has a globally unique identifier that identifies a particular instance of a carousel, called the Carousel NSAP address (Network Service Access Point).

Table 4.8: DVB Carousel NSAP Address syntax

Syntax	bits	Type	Value	Comment
DVBcarouselNSAPaddress()				
AFI	8	uimsbf	0x00	NSAP for private use
Type	8	uimsbf	0x00	Object carousel NSAP Address.
carouselId	32	uimsbf	+	
specifierType	8	uimsbf	0x01	IEEE OUI
specifierData { IEEE OUI }	24	uimsbf	0 x < DVB >	Constant for DVB OUI
dvb_service_location () {				
transport_stream_id	16	uimsbf	+	
original_network_id	16	uimsbf	+	
service_id	16	uimsbf	+	(= MPEG2 program_number)
reserved	32	bslbf	0xFFFFFFFF	
}				
}				

The semantics of the AFI, type, carouselId, specifierData, transport_stream_id, original_network_id, and service_id, and fields are as defined in EN 301 192 [1].

4.7.4 BIOP Messages

4.7.4.1 Directory

The BIOP::DirectoryMessageBody structure consists of a loop of Bindings. A binding correlates an object name (i.e. bindingName) to an IOR and provides additional information about the object. The IOR has to include the BIOP Profile Body when the referenced object belongs to the Carousel.

Strings shall be terminated by the character "0x0"

The BIOP Directory message is an instantiation of the generic object message format.

Table 4.9: BIOP::DirectoryMessage syntax

Syntax	bits	Type	Value	Comment
BIOP::DirectoryMessage() {				
magic	4x8	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_byte	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4x8	uimsbf	0x64697200	”dir” type_id alias
objectInfo_length	16	uimsbf	N2	objectInfo
for (i = 0; i < N2; i++) {				
objectInfo_data_byte	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N3	serviceContextList
for (i = 0; i < N3; i++) {				
serviceContextList_data_byte	8	uimsbf	+	
}				
messageBody_length	32	uimsbf	*	
bindings_count	16	uimsbf	N4	
for (i = 0; i < N4; i++) {				Binding
BIOP::Name(){				
nameComponents_count	8	uimsbf	N5	
for (i = 0; i < N5; i++) {				
id_length	8	uimsbf	N6	NameComponent id
for (j = 0; j < N6; j++) {				
id_data_byte	8	uimsbf	+	
}				
kind_length	8	uimsbf	N7	NameComponent kind
for (j = 0; j < N7; j++) {				
kind_data_byte	8	uimsbf	+	as type_id (see Table 4.4)
}				
}				
}				
bindingType	8	uimsbf	+	0x01 for nobject 0x02 for ncontext
IOP::IOR()			+	objectRef see Table 4.3
objectInfo_length	16	uimsbf	N8	
for (j = 0; j < N8; j++) {				
objectInfo_data_byte	8	uimsbf	+	
}				
}				
}				

The semantics of the fields of the BIOP::DirectoryMessageBody are defined below:

The **byte_order** field indicates the byte ordering used for the following subsequent elements of the message (including message_size). A value of FALSE (0) indicates big-endian byte ordering, and TRUE (1) indicates little endian ordering.

DVB Guideline: *The byte_order field shall have the value of 0x00 meaning that following data is encoded using big-endian byte ordering.*

The **objectKey** field identifies the object that is conveyed in this message. It is identical to the objectKey that is present in the BIOP::ObjectLocation component of the IOR of the object. The value of the objectKey is only

meaningful to the Broadcast Server and is not interpreted by the Client. It will however be used by the Client for a byte by byte comparison to compare this objectKey with the objectKey from an IOR.

DVB Guideline: *The value of the objectKey length field shall be less than or equal to 0x04.*

The **objectKind** field identifies the kind of the object that is conveyed in this message. It is identical to the type_id string that is present in the IOR of the object (see Table 4.4). The value of the objectKind defines the syntax and semantics of the objectInfo field and the messageBody field.

DVB Guideline: *The objectKind of a Directory message shall be "dir".*

The **objectInfo** field contains some or all of the attributes of this object. The syntax and semantics of this field are dependent of the value of the objectKind field.

The **serviceContextList** may be used to pass user private data related to the object interfaces. Its use will not be defined by the present document.

DVB Guideline: *DVB compliant receivers shall be able to skip over the ServiceContextList field.*

The **bindingName** field (i.e. id and kind) contains the *path* specification of the object (as defined by CosNaming).

DVB Guideline: *The BIOP::Name the name shall contain exactly one NameComponent thus* nameComponents_count shall be set to 1.

The **bindingType** field indicates the type of the object binding. Binding can either be of type "ncontext" when the name is bound to a Directory or ServiceGateway object or "nobject" when the name is bound to an object other than Directory or ServiceGateway.

BindingType "composite" is not supported for U-U Object Carousels.

The **objectRef** field contains the IOR of the object.

The **objectInfo** field may contain some of the attributes of the bound object as well as user private information about the object. If attributes of the bound object are carried in this field they shall be the first structures that are encapsulated in this field.

DVB Guideline: *DVB compliant receivers shall be able to skip over information in the objectInfo field.*

4.7.4.2 File

The **FileMessageBody** contains the file data as an octet stream.

Table 4.10: BIOP::FileMessage syntax

Syntax	bits	Type	Value	Comment
BIOP::FileMessage() {				
Magic	4x8	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_byte	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4x8	uimsbf	0x66696C00	”fil” type_id alias
objectInfo_length	16	uimsbf	N2	
DSM::File::ContentSize	64	uimsbf	+	objectInfo
for (i = 0; i < N2 - 8; i++) {				
objectInfo_data_byte	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N3	serviceContextList
for (i = 0; i < N3; i++) {				
serviceContextList_data_byte	8	uimsbf	+	
}				
messageBody_length	32	uimsbf	*	
content_length	32	uimsbf	N4	
for (i = 0; i < N4; i++) {				
content_data_byte	8	uimsbf	+	actual file content
}				
}				

The semantics of the fields of the BIOP::File message are identical as for the BIOP::Directory message except the following rules:

The **objectKind** field identifies the kind of the object that is conveyed in this message. It is identical to the type_id string that is present in the IOR of the object (see Table 4.4). The value of the objectKind defines the syntax and semantics of the objectInfo field and the messageBody field.

DVB Guideline: *The objectKind of a File message shall be "fil".*

4.7.4.3 Stream

DVB Guideline: The objectKind of a Stream message shall be "str".

The BIOP::StreamMessageBody consists a sequence of Taps that are associated with the stream object.

Table 4.11: BIOP::StreamMessage syntax

Syntax	bits	Type	Value	Comment
BIOP::StreamMessage() {				
magic	4x8	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_byte	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	32	uimsbf	0x73747200	”str” type_id alias
objectInfo_length	16	uimsbf	N6	
DSM::Stream::Info_T {				objectInfo
aDescription_length	8	uimsbf	N2	aDescription
for (i = 0; i < N2; i++) {				
aDescription_bytes	8	uimsbf	+	
}				
duration.aSeconds	32	simsbf	+	AppNPT seconds
duration.aMicroSeconds	32	uimsbf	+	AppNPT micro seconds
audio	8	uimsbf	+	
video	8	uimsbf	+	
data	8	uimsbf	+	
}				
for (i = 0; i = N6 - (N2 + 10); i++) {				
objectInfo_byte	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N3	serviceContextList
for (i = 0; i < N3; i++) {				
serviceContextList_data_byte	8	uimsbf	+	
}				
messageBody_length	32	uimsbf	*	
taps_count	8	uimsbf	N4	
for (i = 0; i < N4; i++) {				
id	16	uimsbf	0x0000	undefined
use	16	uimsbf	+	see Table 4.12
association_tag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	no selector
}				
}				

The **stream** field contains one or more Taps that are associated with this stream object. Regarding the content of the stream either one or more Taps are present with a TapUse value of BIOP_ES_USE or one Tap is present with a TapUse value of BIOP_PROGRAM_USE. In the first case, the stream consists of a number of elementary streams, each elementary stream is identified by a BIOP_ES_USE Tap. In the second case the stream consists of an MPEG2 Program, identified by a BIOP_PROGRAM_USE Tap.

The semantics of the fields of a Tap that points to an elementary stream are described below:

- The use field indicates the use of the Tap. The value of this field shall be BIOP_ES_USE.
- The association_tag identifies the broadcast Elementary Stream.
- The selector field shall be empty.

The semantics of the fields of a Tap that points to an MPEG2 Program are described below:

- The use field indicates the use of the Tap. The value of this field shall be BIOP_PROGRAM_USE.
- The association_tag identifies the MPEG2 Program Map Table (PMT) that describes the broadcast program. The association_tag value will correspond with an association_tag value in a deferred_association_tags_descriptor, that points to the PMT (see Subclause 4.7.7.4 Deferred association tags descriptor).
- The selector field shall be empty.

NOTE: The Taps in a stream may also refer to NPT (Normal Play Time), status and event elementary streams.

Table 4.12: Allowed Tap use definitions for Taps in a BIOP::StreamMessage

TapUse field	Value	Broadcast on PID
STR_NPT_USE	0x000B	Stream NPT Descriptors
STR_STATUS_AND_EVENT_USE	0x000C	Both Stream Mode and Stream Event Descriptors
STR_EVENT_USE	0x000D	Stream Event Descriptors
STR_STATUS_USE	0x000E	Stream Mode Descriptors
BIOP_ES_USE	0x0018	Elementary Stream (Video/Audio)
BIOP_PROGRAM_USE	0x0019	Program (DVB Service) Reference

4.7.4.4 Service Gateway

The syntax and semantics of the Service Gateway message are identical to the syntax and semantics of the BIOP::Directory message except the following:

DVB Guideline: *The objectKind of a ServiceGateway message shall be "srg".*

4.7.4.5 StreamEvent

Table 4.13: BIOP::StreamEventMessage syntax

Syntax	bits	Type	Value	Comment
BIOP::StreamEventMessage() {				
magic	4x8	uimsbf	0x42494F50	”BIOP”
version.major	8	uimsbf	0x01	BIOP major version 1
version.minor	8	uimsbf	0x00	BIOP minor version 0
byte_order	8	uimsbf	0x00	big endian byte ordering
message_type	8	uimsbf	*	
message_size	32	uimsbf	*	
objectKey_length	8	uimsbf	N1	
for (i = 0; i < N1; i++) {				
objectKey_data_byte	8	uimsbf	+	
}				
objectKind_length	32	uimsbf	0x00000004	
objectKind_data	4x8	uimsbf	0x73746500	”ste” type_id alias
objectInfo_length	16	uimsbf	N6	
DSM::Stream::Info_T {				
aDescription_length	8	uimsbf	N2	aDescription
for (i = 0; i < N2; i++) {				
aDescription_bytes	8	uimsbf	+	see BIOP::StreamMessage()
}				
duration.aSeconds	32	simsbf	+	see BIOP::StreamMessage()
duration.aMicroSeconds	16	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	N3	
for (i = 0; i < N3; i++) {				
eventName_length	8	uimsbf	N4	
for (j = 0; j < N4; j++) {				
eventName_data_byte	8	uimsbf	+	(including zero terminator)
}				
}				
}				
for (i = 0; i = N6 - (N2 + 10) - (2 + N3 + sum(N4)); i++) {	8	uimsbf	+	
objectInfo_byte				
}				
serviceContextList_count	8	uimsbf	0x00	Empty serviceContextList
for (i = 0; i < N3; i++) {				
serviceContextList_data_byte	8	uimsbf	+	
}				
messageBody_length	32	uimsbf	*	
taps_count	8	uimsbf	N5	
for (i = 0; i < N5; i++) {				
id	16	uimsbf	0x0000	undefined
use	16	uimsbf	+	see Table 4.12
association_tag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	no selector
}				
eventIds_count	8	uimsbf	N3	(= eventNames_count)
for (i = 0; i < N3; i++) {				
eventId	16	uimsbf	+	
}				
}				

DVB Guideline: *The objectKind of a StreamEvent message shall be "ste".*

The **eventIdList** contains the eventIds that are correlated to the event names published in the EvenList_T attribute. The sequence count of the eventIds shall be equal to the sequence count of the EventNames.

NOTE: DSM-CC events do not correspond to DVB-SI events.

4.7.5 Download Data Carousel Messages

4.7.5.1 DownloadInfoIndication

The delivery parameters of the module in the broadcast network are conveyed in a DownloadInfoIndication() message (ISO/IEC 13818-6 [4]). One DownloadInfoIndication() message can convey the module delivery parameters of multiple Modules of the same U-U Object Carousel. The following semantics apply to the fields of the DownloadInfoIndication() message:

The **transactionId** field shall have the same value as the **transactionId** value encapsulated in the selector of the BIOP_DELIVERY_PARA_USE Taps of the IORs of the objects that are carried in Modules described in this message.

DVB Guideline: *If any field of the DownloadInfoIndication message changes, its transaction_id shall be incremented by a positive integer value to a new unique value.*

The **downloadId** field shall have the same value as the **downloadId** field of the DownloadDataBlock() messages which carry the Blocks of the Modules described in this message. Consequently, the value of this field shall be equal to the carouselId of the U-U Object Carousel.

The **blockSize** field contains the block size of all the DownloadDataBlock() messages which convey the Blocks of the Modules described in this message.

The **windowSize, ackPeriod, tCDownloadWindow,** and **tCDownloadScenario** fields are not used and are set to zero.

The **compatibilityDescriptor()** field is not used and has a zero length.

The **moduleId, moduleSize,** and **moduleVersion** fields semantics are in ISO/IEC 13818-6 [4], Subclause 7.3.2.

The **moduleInfoLength** field defines the length in bytes of the moduleInfo field for the described module.

The **moduleInfoBytes** field shall contain the BIOP::ModuleInfo structure. The BIOP::ModuleInfo structure provides additional delivery parameters and the Taps that are used to broadcast the Modules in the network. The syntax and semantics of the BIOP::ModuleInfo structure are shown below.

Table 4.14: BiOP:: ModuleinfoMessage syntax

Syntax	bits	Type	Value	Comment
BIOP::ModuleInfo() {				
ModuleTimeOut	32	uimsbf	+	
BlockTimeOut	32	uimsbf	+	
MinBlockTime	32	uimsbf	+	
taps_count	8	uimsbf	N1	
for (j = 0; j < N1; j++) {				
Id	16	uimsbf	0x0000	user private
Use	16	uimsbf	0x0017	BIOP_OBJECT_USE
association_tag	16	uimsbf	+	
selector_length	8	uimsbf	0x00	
}				
UserInfoLength	8	uimsbf	N2	
for (j = 0; j < N2; j++) {				
userInfo_data_byte	8	uimsbf	+	(including zero terminator)
}				
}				

The **moduleTimeOut** field gives the time out value in microseconds that may be used to time out the acquisition of all Blocks of the Module.

The **blockTimeOut** field gives the time out value in microseconds that may be used to time out the reception of the next Block after a Block has been acquired.

The **minBlockTime** field indicates the minimum time period that exists between the delivery of two subsequent Blocks of the described Module. Clients may use this value to adjust their acquisition procedures for optimization purposes.

The **Taps** field of BIOP::ModuleInfo shall contain at least one Tap with the TapUse value of BIOP_OBJECT_USE. This Tap shall point to the network connection on which the Modules are broadcast. The semantics of the fields of this Tap are described in Subclause 4.7.2.5.

The userInfo field of BIOP::ModuleInfo shall be structured as a loop of descriptors which enables the use of Module descriptors as defined in DVB Data Carousels.

DVB Guideline: The receiver shall support especially the compressed_module_descriptor (tag 0x09) used to signal that the module is transmitted in compressed form.

The use of the **privateDataLength** and **privateDataByte** fields is not defined by the present document.

DVB Guideline: DVB compliant receivers shall be able to skip over the private data field.

4.7.5.2 DownloadServerInitate

The IOR of the Service Gateway is broadcast by means of DownloadServerInitiate() messages.

The following semantics apply on the fields of the DownloadServerInitiate() message:

The **serverId** field shall be set to 20 bytes with the value 0xFF. The Carousel Specifier is defined below.

The **compatibilityDescriptor()** field is not used and has a zero length.

The **privateDataLength** field of the DownloadServerInitiate() message defines the length in bytes of the privateDataByte fields that follow this field.

The data in the **privateDataByte** field of the DownloadServerInitiate() message shall contain the BIOP::ServiceGatewayInfo structure. The syntax and semantics of the BIOP::ServiceGatewayInfo structure are defined below:

Table 4.15: ServiceGatewayInfo() syntax

Syntax	bits	Type	Value	Comment
ServiceGatewayInfo () {				
IOP::IOR()			+	see Table 4.3
downloadTaps_count	8	uimsbf	N1	software download Taps
for (i = 0; i < N1; i++) {				
Tap()	8	uimsbf	+	
}				
serviceContextList_count	8	uimsbf	N2	serviceContextList
for (i = 0; i < N2; i++) {				
serviceContextList_data_byte	8	uimsbf	+	
}				
userInfoLength	16	uimsbf	N3	user info
for (i = 0; i < N3; i++) {				
userInfo_data_byte	8	uimsbf	+	
}				
}				

The **objectRef** field contains the IOR of the ServiceGateway.

The semantics of the **Taps field** and s**erviceContextList** is not defined in the present document.

The **user info** field shall be structured as a descriptor loop. The descriptors in this loop shall be either descriptors as defined in the DVB Data Broadcasting Specification or private descriptors.

4.7.5.3 DownloadDataBlock

The DownloadData Message is defined in ISO/IEC 13818-6 [4]. The use of the fields is defined in the DSM-CC specification ISO/IEC 13818-6 [4].

4.7.6 MPEG2 Sections

ETS 300 802 [2] defines a private_section structure which DSM-CC uses to provide re-assembly of Transport Stream Packets into DSM-CC messages. DSM-CC defines additional semantics on private_sections to support additional DSM-CC requirements. Called DSMCC_section, the structure is compatible with the private_section syntax so that compliant MPEG2 Systems decoders may be used. The DSM-CC_section syntax is defined in ISO/IEC 13818-6 [4].

DVB Guideline: *The encoding of the table_id_extension, version_number, section_number, and last_section_number are defined in Table 4.16.*

Table 4.16: Encoding of DSMCC_section fields

Message	table_ id	table_id_extension	version_ number	section_ number	last_section_ number
Download-ServerInitiate (DSI)	0x3B	two LSB bytes of transaction_id of DSI	0x00	0x00	0x00
Download-InfoIndication (DII)	0x3B	two LSB bytes of transaction_id of DII	0x00	0x00	0x00
Download-DataBlock (DDB)	0x3C	moduleId	module Version % 32	blockNumber % 256	Max(section_ number)

DVB Guideline: *For DownloadServerInitiate messages the 2 least significant bytes of the transaction_id shall be in the range 0x0000 - 0x0001.*

DVB Guideline: *DownloadInfoIndication messages the 2 least significant bytes of the transaction_id shall be in the range 0x0002 - 0xFFFF.*

DVB Guideline: *DVB has put some limitations to the basic DSM-CC specification regarding the transaction_id field to allow for easy filtering options to customer decoders. In particular, DSI messages have a value of 0x0000 or*

0x0001 for the two LSB bytes. This enables receivers to bootstrap the carousel by setting up the section filters for table_id = 0x3B (DownloadControlMessages) and table_id_extension = 0x0000 or 0x0001. Once the DSI message has been acquired the receiver can set up the section filter to listen to the other value of the two LSB bytes of the transaction_id. This shall trigger the receiver immediately once the carousel content is being updated.

4.7.7 Use of PSI descriptors

The Object Carousel specification in ISO/IEC 13818-6 [4] is network independent and is applicable for any type of Broadcast Network. Network independence is achieved by using the Tap concept. A Tap facilitates a reference to a particular network connection by means of an association tag. In the course of resolving an object, Clients have to associate the Taps to broadcast connections of the network. Clients need therefore an association table that makes the associations between the Taps and the connections of the broadcast network.

For the implementation of U-U Object Carousels on top of Broadcast Networks that are based on MPEG2 Transport Streams, the PSI mechanisms facilitate:

1. the association of a MPEG2 Program (i.e. PMT) with an Object Carousel;
2. the association of a Tap with a PID or a MPEG2 Program;
3. the localization of the PID on which the IOR of the Service Gateway is broadcast; and
4. the distributed implementation of an Object Carousel on top of multiple MPEG2 Programs.

This subclause explains the use of three MPEG2 descriptors (ISO/IEC 13818-6 [4]) that provide this functionality (See also ISO/IEC 13818-6 [4]).

4.7.7.1 Carousel identifier descriptor

The carousel identifier descriptor facilitates the association between a MPEG2 Program and an Object Carousel. The syntax and semantics of the carousel_identifier_descriptor() are described below (see ISO/IEC 13818-6 [4]).

This optional mechanism allows to to acquire the ServiceGateWay of a ServiceDomain without first loading the Download Server Initiate and Download Indication Information messages.

Table 4.17: carousel_identifier_descriptor

Syntax	bits	Type	Value	Comment
carousel_identifier_descriptor () {				
descriptor_tag	8	uimsbf	0x13	
descriptor_length	8	uimsbf	*	
carousel_id	32	uimsbf	+	
FormatId	8	uimsbf		Registered Identifier of the FormatSpecifier
FormatSpecifier(){				
FormatSpecifier_byte	8	uimsbf		see Table 4.17a N2 bytes
}				
for (i = 0;i < N1;i++){				
private_data_byte	8	uimsbf		
}				
}				

DVB Guideline: *The carousel_identifier_descriptor() shall be inserted in the second descriptor loop of the PMT (ES_info) corresponding to the elementary stream carrying the DSI of the object carousel. This allows more than one object carousel per MPEG-program and implicitly identifies the PID on which each carousel should be booted from.*

The insertion of a carousel_identifier_descriptor() is also necessary to support the use of the DVBcarouselNSAPaddress, such as in the resolution of a LiteOptionsProfileBody reference.

The **FormatId** identifies the format of a FormatSpecifier carried in the private data field of the descriptor. The syntax and semantics of this structure are defined below:

Table 4.17a: FormatSpecifier in the carousel_identifier_descriptor

FormatId Value	Format Specifier Definition	length [bits]	Comment
0x00	no FormatSpecifier		A value of 0x00 indicates the absence of a FormatSpecifier. Thus the location of the ServiceGateway is only possible through the "standard" way interpreting the DSI and DII messages.

0x01	FormatSpecifier{
	ModuleVersion
	ModuleId
	BlockSize
	ModuleSize
	CompressionMethod
	OriginalSize
	TimeOut
	ObjectKeyLength
	for (i = 0;i < N1;i++{
	ObjectKeyData
	}
	}
		
		8
		16
		16
		32
		8
		32
		8
		8
		
		8
			This FormatSpecifier is an aggregation of the fields necessary to locate the ServiceGateway, also found in the DSI and DII messages.
			
			NOTE: All field types are "uimsbf".
			
			Timeout in seconds.
			
			Object key of the service gateway object.

0x02...0x7F	reserved for future use		The format Id values from 0x02 to 0x7F are reserved for future use of DVB.
0x80...0xFF	reserved for private use		The format Id values from 0x80 to 0xFF are reserved for private use.

FormatId 0x01 identifies that the FormatSpecifier contains information (also found in the DSI and DII messages) that can be used to locate the ServiceGateway of the object carousel. Supporting this FormatID may have consequences for the broadcast server since this information has to be kept consistent with changes to the ServiceGateway object and the module in which it is delivered.

DVB Guideline: The presence of the FormatSpecifier with FormatId 0x01 implies that the DSI message and the module containing the ServiceGateway are carried on the same PID.

4.7.7.2 Association tag descriptor

The association_tag_descriptor (ISO/IEC 13818-6 [4]) facilitates the association between an association_tag and a PID and is therefore similar as the stream_identifier descriptor of DVB SI (EN 300 468 [6]). The assocation_tag descriptor uses however 16-bit assocation_tag (as opposed to the 8-bit component_tag of the stream_identifer_descriptor) and facilitates the identification of the PID on which the ServiceGateway is broadcast. The latter function allows receivers to bootstrap the Object Carousel efficiently from a PMT with a large number of PIDs. To label a PID with a particular association_tag value, the Server shall insert the association_tag descriptor in the descriptor loop of that PID.

The syntax and semantics of the association_tag_descriptor are described below:

Table 4.18: association_tag_descriptor

Syntax	bits	Type	Value	Comment
association_tag_descriptor () {				
descriptor_tag	8	uimsbf	0x14	
descriptor_length	8	uimsbf	*	
association_tag	16	uimsbf	+	
use	16	uimsbf	0x0000 0x0100 - 0x1FFF 0x2000 - 0xFFFF	DSI with IOR of SGW DVB reserved user private
if (use == 0x0000) {				
selector_length	8	uimsbf	0x08	
transaction_id	32	uimsbf	+	transaction_id of DSI
timeout	32	uimsbf	+	timeout for DSI
} else if (use == 0x0001) {				
selector_length	8	uimsbf	0x00	

Syntax	bits	Type	Value	Comment
} else {				
selector_length	8	uimsbf	N1	
for (i = 0; i < N1; i++) {				
selector_byte	8	uimsbf		
}				
}				
for (i = 0; i < N2; i++) {				
private_data_byte	8	uimsbf		private data
}				
}				

The **use** field may indicate the usage of the PID and shall specify the syntax and semantics of the selector field. If the use value equals 0x0000 then the DownloadServerInitiate message that carries the IOR of the Service Gateway is broadcast on this PID. In this case the data in the selector_byte fields shall contain the transaction_id and a timeout value.

The semantics of the transaction_id and timeout fields are as follows.

The value of the **transaction_id** field shall correspond to the transaction_id of the DownloadServerInitiate() message that conveys the IOR of the Service Gateway of the U-U Object Carousel. Except when the transaction_id in the association_tag_descriptor has the value of 0xFFFFFFFF. This value indicates that the transaction_id of the DownloadServerInitiate() message is not known at this point, but all DownloadServerInitiate() messages broadcast on the identified PID are valid. A transaction_id value of 0xFFFFFFFF may be used when the content of the DownloadServerInitiate() message is allowed to change (and thus the transaction_id in the message changes), without the need to update the PMT that contains the association_tag_descriptor.

The **timeout** field shall indicate the time-out period in microseconds that may be used to time out the acquisition of the DownloadServerInitiate() message. A special value of the timeout (0xFFFFFFFF) indicates that no timeout value is known at this point. Allowing a "static" PMT as described above.

DVB Guideline: *The default value for the use field shall be 0x0100. This means that the associated PID may or may not broadcast a DSI message.*

DVB Guideline: *DVB reserves the range of 0x0101 to 0x01FF for the use field for future use.*

4.7.7.3 Stream identifier descriptor

The stream_identifier_descriptor [DVB-SI] facilitates the association between a component_tag and a PID in an efficient way and may be used instead of (or in combination with) the association_tag descriptors. However since the component_tag field of a stream_identifier_descriptor is only an 8-bit field a mapping is necessary between component_tags and assocation_tags.

DVB Guideline: *A stream_identifier_descriptor in the descriptor loop of a PID shall be equivalent with an association_tag_descriptor for that PID with an association_tag value of LSB = <component_tag> and a use value of 0x0100.*

NOTE: This matching provides the flexibility to distribute the object carousel over multiple elementary streams and still use the same component_tag value in the different PMTs to refer to this particular data broadcast service.

4.7.7.4 Deferred association tags descriptor

An Object Carousel may use multiple PIDs, Services, and Transport Streams to broadcast the objects and associated control information. To facilitate Clients with the localization of all association_tags that are used in the different MPEG2 Programs for the Object Carousel, a descriptor is defined that may be inserted in the first descriptor loop of the PMTs of the MPEG2 Programs that implement the Object Carousel. The deferred_assocation_tags_descriptor contains association_tags that are used within the Object Carousel but that are not associated with a PID in the PMT in which the descriptor resides. The deferred_association_tags_descriptor contains therefore a forward reference to an

MPEG2 Program that does contain the PID to which the association tag is linked. Multiple deferred association tags descriptors may be inserted in a PMT if necessary.

In addition a deferred_association_tag_descriptor may be used to refer to another DVB service (MPEG2 program) as a result of a BIOP_PROGRAM_USE Tap.

Deferred_association_tags should be used whenever an object carrousel is broadcast using multiple services. For every service that carries a part of the carrousel, the list of deferred association_tags must be complete to avoid failing or false mapping of association_tags.

The syntax and semantics of the deferred_association_tags_descriptor() are described below:

Table 4.19: deferred_association_tags_descriptor

Syntax	bits	Type	Value	Comment
deferred_association_tags_descriptor () {				
descriptor_tag	8	uimsbf	0x15	
descriptor_length	8	uimsbf	*	
association_tags_loop_length	8	uimsbf	2*N1	length in bytes
for (n = 0; n < N1 ; n++) {				
association_tag	16	uimsbf	+	
}				
transport_stream_id	16	uimsbf	+	
program_number	16	uimsbf	+	
org_network_id	16	uimsbf	+	
for (n = 0; n < N ; n++) {				
Private_data_byte	8	uimsbf	+	
}				
}				

4.7.8 Information in the SI and PSI

For signalling just the use of the DVB object carousel the data_broadcast id shall be set to 0x0007.

NOTE: If the use of the object carousel forms part of a specification which has registered a data_broadcast id, this alternative value (with the appropriate syntax for the selector fields) may be used instead.

4.7.8.1 SI Descriptor

The data_broadcast_descriptor in the SI can be used with the above value to indicate the presence of an DVB object carousel within a Service.

In this case the selector field of the data_broadcast_descriptor contains a loop of object names that allows the bootstrapping of applications within the object carousel. The loop contains an ISO_639_language_code field which can be used (for example) to start an application based on preferred language.

DVB Guideline: *The object names used in the data_broadcast_descriptor shall exist in the Object Carousel.*

4.7.8.2 Descriptors in PSI

The data broadcast_id descriptor can be used in a similar way as for Data Carousels (see Subclause 4.6.7.1)

4.7.9 Assignment and use of transactionId values

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 UK DTT object carousel this should be interpreted as "changed".

An object carousel are carried on top the Data Carousels and may be distributed over multiple Data Carousels. By a Data Carousel used below the object carousel, we mean in this specification 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.

The transactionId has been split up into a number of sub-fields defined in Table 26. 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 receivers.

Table 4.20: 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/changed every time the control message is updated.
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...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 receiver 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.

Annex A (informative): DSM-CC messages for Data Carousel

This informative Annex contains the syntax of the DSM-CC Download messages as defined per July 12 1996. The semantic description of each field indicates where possible the value to use when implementing a DVB Data Carousel using this protocol.

A.1 dsmccMessageHeader

Table A.1: MPEG2 DSM-CC Message Header Format

Syntax	Number of Bytes
dsmccMessageHeader() {	
protocolDiscriminator	**1**
dsmccType	**1**
messageId	**2**
transactionId	**4**
reserved	**1**
adaptationLength	**1**
messageLength	**2**
if(adaptationLength > 0) {	
dsmccAdaptationHeader()	
}	
}	

The **protocolDiscriminator** field is used to indicate that the message is a MPEG2 DSM-CC message. The value of this field shall be 0x11.

[The use of protocolDiscriminator 0x11 is dependent upon the response of ITU-T SG11 and ISO/IEC JTC1 to a liaision letter requesting that this value be assigned to DSM-CC.]

The **dsmccType** field is used to indicate the type of MPEG2 DSM-CC message. The value of this field shall be 0x03 to indicate that the message is a U-N Download message.

The **messageId** field indicates the type of message which is being passed. The values of the messageId are defined within the scope of the dsmccType.

The **transactionId** field is used for session integrity and error processing and shall remain unique for a period of time such that there will be little chance that command sequences collide. The transactionId is of local significance - i.e. the value should be chosen by the broadcast server.

The **reserved** field is ISO/IEC 13818-6 [4] reserved. This field shall be set to 0xFF.

The **adaptationLength** field indicates the total length in bytes of the adaptation header.

The **messageLength** field is used to indicate the total length in bytes of the message following this field. This length includes any adaptation headers indicated in the adaptationLength and the message payload indicated by the messageId field.

A.2 dsmccDownloadDataHeader

Table A.2: DSM-CC Download Data Header Format

Syntax	Number of Bytes
dsmccDownloadDataHeader() {	
ProtocolDiscriminator	**1**
DsmccType	**1**
MessageId	**2**
DownloadId	**4**
Reserved	**1**
AdaptationLength	**1**
MessageLength	**2**
for(adaptationLength > 0) {	
dsmccAdaptationHeader()	
}	
}	

The **protocolDiscriminator** field is used to indicate that the message is a MPEG2 DSM-CC message. The value of this field shall be 0x11.

[The use of protocolDiscriminator 0x11 is dependent upon the response of ITU-T SG11 and ISO/IEC JTC1 to a liaision letter requesting that this value be assigned to DSM-CC.]

The **dsmccType** field is used to indicate the type of MPEG2 DSM-CC message. The value of this field shall be 0x03 to indicate that the message is a U-N Download message.

The **messageId** field indicates the type of message which is being passed. The values of the messageId are defined within the scope of the dsmccType.

The **downloadId** field is used to associate the download data messages and the download control messages of a single instance of a download scenario.

The **reserved** field is ISO/IEC 13818-6 [4] reserved. This field shall be set to 0xFF.

The **adaptationLength** indicates the total length in bytes of the adaptation header.

The **messageLength** field is used to indicate the total length in bytes of the message following this field. This length includes any adaptation headers indicated in the adaptationLength and the message payload indicated by the messageId field.

A.3 DownloadServerInitiate

Table A.3: DownloadServerInitiate message

Syntax	Number of Bytes
DownloadServerInitiate() {	
dsmccMessageHeader()	
serverId	**20**
compatibilityDescriptor()	
privateDataLength	**2**
for(i = 0;i < privateDataLength;i++) {	
privateDataByte	**1**
}	
}	

The **serverId:** field shall be set to 20 bytes with the value 0xFF (i.e. the field is not used).

The **compatibilityDescriptor()** structure shall only contain the compatibilityDescriptorLength field of the compatibilityDescriptor as defined in DSM-CC (ISO/IEC 13818-6 [4]). It shall be set to the value 0x0000 (i.e. the field is not used).

The **privateDataLength** field defines the length in bytes of the following structure.

The **privateDataByte** fields shall convey the GroupInfoIndication structure defined in the DVB Specification for Data Broadcasting EN 301 192 [1].

A.4 DownloadInfoIndication

Table A.4: DownloadInfoIndication message

Syntax	Number of Bytes
DownloadInfoIndication() {	
dsmccMessageHeader()	
downloadId	**4**
blockSize	**2**
windowSize	**1**
ackPeriod	**1**
tCDownloadWindow	**4**
tCDownloadScenario	**4**
compatibilityDescriptor()	
numberOfModules	**2**
for(i = 0;i < numberOfModules;i++) {	
moduleId	**2**
moduleSize	**4**
moduleVersion	**1**
moduleInfoLength	**1**
for(i = 0;i < moduleInfoLength;i++) {	
moduleInfoByte	**1**
}	
}	
privateDataLength	**2**
for(i = 0;i < privateDataLength;i++) {	
privateDataByte	**1**
}	
}	

The **downloadId** field is the identifier of the download scenario in progress. The downloadId shall be uniquely defined within the Network for Data Carousel scenario and unique within the connection for the flow-controlled and non-flow-controlled scenarios. This identifier shall be used in all of the subsequent DownloadDataBlock, DownloadDataRequest, and DownloadCancel messages used by the download scenario in progress.

The **blockSize** field is the length in bytes of the data in every block carried in the DownloadDataBlock messages, except for the last block of each module which may be smaller than blockSize.

The **windowSize** is unused for broadcast Data Carousel scenarios and shall be set to 0.

The **ackPeriod** is unused for broadcast Data Carousel scenarios and shall be set to 0.

The **tCDownloadWindow** is unused for broadcast Data Carousel scenarios and shall be set to 0.

The **tCDownloadScenario** field indicates the time out period in microseconds for the entire download scenario in progress.

The **compatibilityDescriptor()** structure shall only contain the compatibilityDescriptorLength field of the compatibilityDescriptor as defined in DSM-CC, (ISO/IEC 13818-6 [4]). It shall be set to the value 0x0000 (i.e. the field is not used).

The **numberOfModules** field is the number of modules described in the loop following this field. For flow-controlled and non-flow controlled download scenarios, the loop describes all the modules that have to be downloaded by the Client. For the Data Carousel scenario, the loop describes a subset of all the modules associated with this Data Carousel, although it may describes all of them.

The **moduleId** field is an identifier for the module that is described by the moduleSize, moduleVersion, and moduleInfoByte fields. The moduleId is unique within the scope of the downloadId.

The **moduleSize** field is the length in bytes of the described module.

The **moduleVersion** field is the version of the described module.

The **moduleInfoLength** field defines the length in bytes of the moduleInfo field for the described module.

The **moduleInfoByte** fields shall convey a list of descriptors. Each list will define one or more attributes of the associated module.

NOTE: That the interpretation of these fields is different when the moduleId is in the range 0xFFF0 to 0xFFFF. In this case, these fields carry the ModuleInfo structure as defined by DAVIC.

The **privateDataLength** field defines the length in bytes of the following privateDataByte field.

The **privateDataByte** field is user defined.

A.5 DownloadDataBlock

Table A.5: DownloadDataBlock

Syntax	Number of Bytes
DownloadDataBlock() {	
dsmccDownloadDataHeader()	
moduleId	**2**
moduleVersion	**1**
reserved	**1**
blockNumber	**2**
for(i = 0;i < N;i++) {	
blockDataByte	**1**
}	
}	

The **moduleId** field identifies to which module this block belongs.

The **moduleVersion** field identifies the version of the module to which this block belongs.

The **reserved** field is reserved by ISO/IEC 13818-6 [4] and shall be set to 0xFF.

The **blockNumber** field identifies the position of the block within the module. Block number 0 shall be the first block of a module.

The **blockDataByte** conveys the data of the block.

A.6 DownloadCancel

Table A.6: DownloadCancel message

Syntax	Number of Bytes
DownloadCancel() {	
dsmccMessageHeader()	
downloadId	**4**
moduleId	**2**
blockNumber	**2**
downloadCancelReason	**1**
reserved	**1**
privateDataLength	**2**
for(i = 0;i < privateDataLength;i++) {	
privateDataByte	**1**
}	
}	

The **downloadId** field is the identifier of the instance of the download scenario in progress. It shall be used this to associate the DownloadCancel message to a particular download scenario in progress or Data Carousel.

The **moduleId** and **blockNumber** fields indicate the last processed DownloadDataBlock message at the time of the cancel. If no data blocks have been processed, these fields shall be set to 0.

The **downloadCancelReason** field contains a reason code that explains the reason for the cancellation.

The **reserved** field is reserved by ISO/IEC 13818-6 [4] and shall be set to 0xFF.

The **privateDataLength** field defines the length in bytes of the following privateDataByte fields.

The use of the **privateDataByte** field is not specified by the DVB Data Carousel and may be used for proprietary information.

Annex B (informative): Encapsulation of DSM-CC messages in MPEG2 sections

This informative Annex illustrates how DSM-CC messages are encapsulated in MPEG2 sections. Please refer to the DSM-CC specification for the precise semantics.

When DSM-CC Download messages are encapsulated in MPEG2 Transport Streams, the DSMCC_section syntax shall be used. This structure inherits all of the Private_section syntax as defined in ISO/IEC 13818-1 [3]. Special semantics apply to the encoding of particular fields in the DSMCC_section header. The mapping of the DSMCC_section into MPEG2 Transport Stream Packets and the maximum length of a DSMCC_section are governed by the semantics for Private_sections defined in ISO/IEC 13818-1 [3].

In some implementations, it is desirable to use the CRC_32 available in Private_sections. Because some systems may have difficulty calculating a CRC_32, the DSMCC_section syntax defines an alternative to using CRC_32. To be consistent with ISO/IEC 13818-1 [3], if the section_syntax_indicator is set to "1", then the CRC_32 shall be present and correct. In the case where the section_syntax_indicator is "0", the syntax of the section is the same as when the section_syntax_indicator is "1", except that the CRC_32 field is replaced with the checksum field. The resultant syntax is still compliant to ISO/IEC 13818-1 [3], since the payload following the section_length field shall be treated as private data.

Since the section_syntax_indicator bit itself may be subject to a bit error, the private_indicator field shall be set to the complement of the section_syntax_indicator value. If the section_syntax_indicator is "0", then the private_indicator shall be verified to be "1", and if it is not, the section has suffered an error. Similarly, if the section_syntax_indicator is "1" then private_indicator shall be "0".

When section_syntax_indicator is "0" (CRC is not used) and the checksum field has been set to 0, another form of error detection shall be provided at a different layer. This requirement is imposed to ensure the DSMCC_section maintains the minimal requirements the present document imposes on its transport protocol.

For syntax and semantics related to the carriage of private_sections (and therefore DSMCC_sections) within the MPEG Transport Stream, see ISO/IEC 13818-1 [3] Subclause 2.4.4 Program specific information. This includes the setting of the payload_unit_start_indicator, the presence of the pointer_field in the Transport Stream packet payload, and the use of packet stuffing bytes.

Unless otherwise restricted, DSM-CC tables (i.e., one or more DSMCC_sections with the same table_id) may be contained in Transport Stream packets with the same value PID as other private_section formatted tables (for example, in ISO/IEC 13818-1 [3] stream_type 0x05), if table_id parsing is done.

When DownloadDataBlock messages are carried in MPEG2 Transport Streams, only DownloadDataBlock messages with the same value of **downloadId** shall be contained in Transport Stream packets with the same value PID. This means that each PID can only deliver download data messages from a single Data Carousel. There is no such restriction specified for downloadcontrol messages, allowing such messages from any number of Data Carousels to be transported in the same elementary stream. In these cases the **transactionId** of a particular top-level control message has to be explicitly identified using the data_broadcast_descriptor in SI to achieve predictable behaviour.

Table B.1: DSM-CC Section Format

Syntax	Number of bits	Mnemonic
DSMCC_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
private_indicator	**1**	**bslbf**
reserved	**2**	**bslbf**
dsmcc_section_length	**12**	**uimsbf**
table_id_extension	**16**	**uimsbf**
reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
if(table_id == 0x3A) {		
LLCSNAP()		
}		
else if (table_id == 0x3B) {		
userNetworkMessage()		
}		
else if (table_id == 0x3C) {		
downloadDataMessage()		
}		
else if (table_id == 0x3D) {		
DSMCC_descriptor_list()		
}		
else if (table_id == 0x3E) {		
for (i = 0;i < dsmcc_section_length - 9;i++) {		
private_data_byte		
}		
}		
if(section_syntax_indicator == "0") {		
checksum	**32**	**uimsbf**
}		
else {		
CRC_32	**32**	**rpchof**
}		
}		

NOTE 1: The DownloadServerInitiate message, the DownloadInfoIndication message, and the DownloadCancel message are in the userNetworkMessage class.

NOTE 2: The DownloadDataBlock message is within the downloadMessage class.

Annex C (informative): Naming of objects in directories

In DSM-CC, the Directory objects provide a hierarchical tree-like directory structure (actually, the directory structure can be even more general graph than a tree). Each Directory object may contain references to other Directories (i.e. subdirectories) and other objects. When a Object is bound to a Directory, a name string is assigned to it that uniquely identifies the object within that directory. The ServiceGateway object is the root directory of the directory hierarchy. The path that includes the names starting from the ServiceGateway via possible subdirectories to an object identifies that object uniquely within the object carousel. An object can possibly be bound to multiple directories and thus have many paths all pointing to the same object.

The following conventions for delimiting names and directories are the following:

a) The forward slash "/" shall be used as a delimiter between directory names and object names.

b) The forward slash is not allowed as part of a name.

Relative path names are not required therefore no other convention is necessary.

For various reasons, DSM-CC and the object carousels use many slightly different data structures for storing the path in different contexts. Thus, in different contexts the data structures that are used for storing the path may be different while they are still referring to the same path, i.e. name strings in the data structures are the same.

C.1 Data structures used for names in DSM-CC User-to-User API

The User-to-User API uses two different data structures for the path in different contexts. The reason for this is that the Directory object of DSM-CC inherits from the CosNaming::NamingContext object of CORBA. DSM-CC however adds some functionality to that and for this additional functionality it has been necessary to define a separate data structure for passing the path information.

The CosNaming::NameComponent is the basis for all name data structures. It represents one part of the whole path name to the object, i.e. the name within one subdirectory. The NameComponent structure contains two fields: id and kind. The id field contains the actual name string and the kind field contains the type of the object.

The CosNaming::Name is a sequence of NameComponets and represents the whole path. However, this structure normally identifies the relative path starting from the directory where it is used in. When the Name is used in the ServiceGateway, it naturally represents the full absolute path.

In some functions, the Name is carried inside a structure called CosNaming::Binding. In addition to the Name, the Binding contains a field that identifies the BindingType. The purpose of the BindingType is to identify a classification of the object that the Name points to.

DSM-CC has defined another data structure for the path, the DSM::PathSpec. The PathSpec consists of a sequence of DSM::Step structures. The Step contains the same NameComponent as is used in the CosNaming::Name and also an additional process flag that is used in some functions to inform if the operation should be applied to this part of the path or not. When the PathSpec is used in DSM-CC, there is usually another parameter also: a PathType. The PathType identifies the way how the PathSpec should be interpreted. It differentiates between the two different ways how the PathSpec is used. When the PathType is DEPTH, the meaning of the PathSpec is equivalent to the Name, i.e. it is a relative path down the directory hierarchy starting from the current directory. However, when the PathType is BREADTH, the NameComponents in the PathSpec are used to identify multiple different objects within the same directory.

C.2 Data structures used for names in Object Carousels

For optimizing the transportation, the Object Carousels use slightly different data structures than the U-U API. These data structures are however intended to be equivalent with the ones that are used in the API.

The BIOP::NameComponent is equivalent to the CosNaming::NameComponent, but the maximum lengths for the strings have been added to optimize the encoding.

The BIOP::Name is equivalent to the CosNaming::Name, but it defines an upper bound for the number of NameComponents in it to optimize the encoding.

The DirectoryMessage of the Object Carousels provides the necessary information for implementing the Directory object. The Directory message contains BIOP::Bindings that include the Name that identifies the path to the object starting from this directory and the Interoperable Object Reference that contains the necessary information to locate the actual object. The BIOP::Binding is different from the CosNaming::Binding so that the BIOP::Binding contains the object reference while the CosNaming::Binding does not. This is because in the object carousels, it is used to carry the location of the object, while in the API the location of the object is not visible to the application but internal to the Directory object.

C.3 CORBA strings in Object Carousels

In a number of places Object Carousel messages include text strings. These are all formatted in accordance with Subclause 12.3.2 of CORBA V2.0. I.e. the text is preceeded by a 32-bit integer specifying the length of the string and followed by a null terminator. In general this is can be seen clearly in the syntax tables that follow. However, for clarity CORBA format strings are used in the the following places:

Table C.1: Location of CORBA format strings

string	location
objectKind_data	BIOP::FileMessage syntax
objectKind_data, id_data, kind_data	BIOP::DirectoryMessage syntax
objectKind_data	BIOP::StreamMessage syntax
objectKind_data, eventName_data	BIOP::StreamEventMessage syntax
type_id_byte	BOP::IOR syntax
id_data, kind_data	Syntax of Options Profile Body with ServiceLocation component

Annex D (informative): Example of an Object Carousel

Figure D.1 illustrates an object carousel that is distributed over three elementary streams belonging to the same service.

The DownloadServerInitiate (DSI) message is carried on the first elementary stream. It contains the object reference that points to the service gateway. 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 service gateway object is in the module number 1 that is carried on the sec-ond elementary stream (indicated by a BIOP_OBJECT_USE tap structure in the DII message).

The Service Gateway 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" subdi-rectories are distributed over the first and second elementary stream.

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.

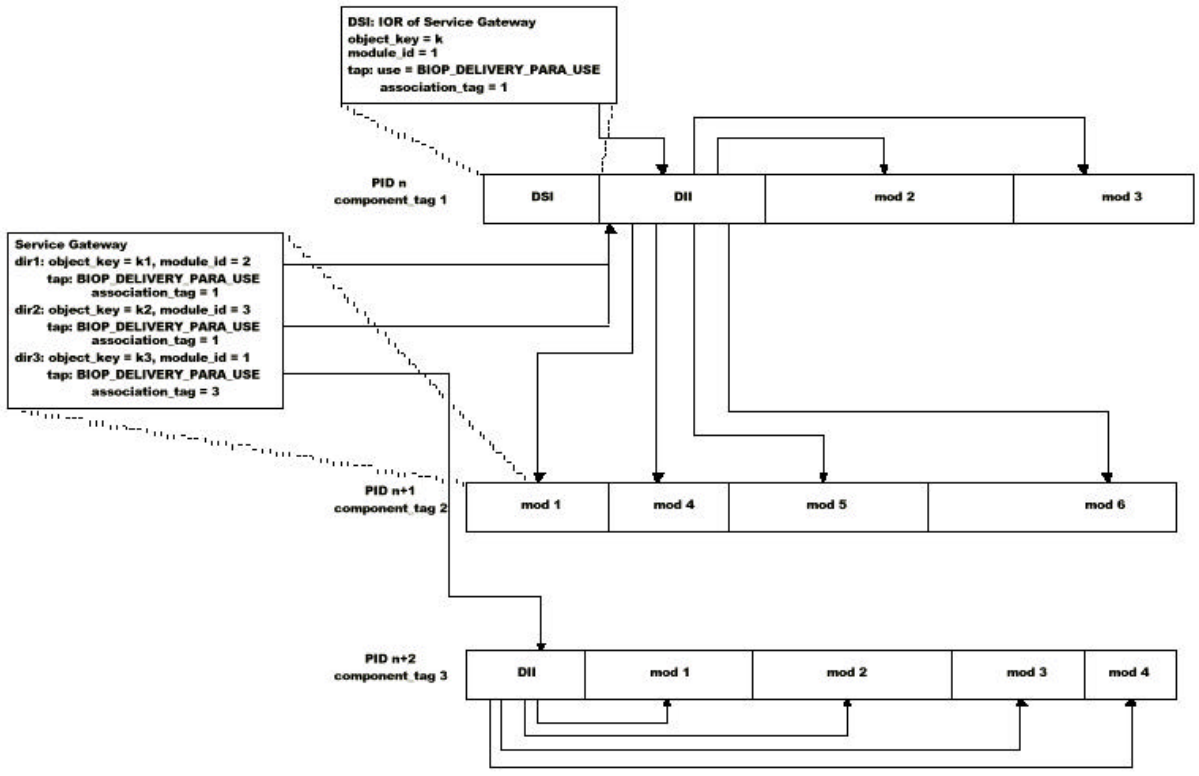


Figure D.1: Object Carousel example

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.

History

Document history		
V1.1.1	February 1999	Publication