Doc. A/91
10 June 2001

ATSC Recommended Practice: Implementation Guidelines for the ATSC Data Broadcast Standard (Doc. A/90)

Advanced Television Systems Committee
1750 K Street, N.W.
Suite 1200
Washington, D.C. 20006
www.atsc.org

The Advanced Television Systems Committee (ATSC), is an international, non-profit membership organization developing voluntary standards for the entire spectrum of advanced television systems.

Specifically, ATSC is working to coordinate television standards among different communications media focusing on digital television, interactive systems, and broadband multimedia communications. ATSC is also developing digital television implementation strategies and presenting educational seminars on the ATSC standards.

ATSC was formed in 1982 by the member organizations of the Joint Committee on InterSociety Coordination (JCIC): the Electronic Industries Association (EIA), the Institute of Electrical and Electronic Engineers (IEEE), the National Association of Broadcasters (NAB), the National Cable Television Association (NCTA), and the Society of Motion Picture and Television Engineers (SMPTE). Currently, there are approximately 190 members representing the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

ATSC Digital TV Standards include digital high definition television (HDTV), standard definition television (SDTV), data broadcasting, multichannel surround-sound audio, and satellite direct-to-home broadcasting.

Table of Contents

1. SCOPE 9

2. REFERENCES 9

3. ACRONYMS AND DEFINITIONS 10

3.1 Acronyms 10

3.2 Definitions 11

3.3 This Document's Table Structure Format 11

4. INTRODUCTION TO THE IMPLEMENTATION GUIDELINES OF THE ATSC A/90 DATA BROADCAST SPECIFICATION 11

5. INTRODUCTION TO THE ATSC A/90 DATA BROADCAST STANDARD 12

5.1 Data Service Definition 15

5.2 Data Encapsulations 15

5.2.1 Packetization of Data Entities 16

5.3 Encapsulation Protocol Selection Guidance 17

5.3.1 Bounded Data Blobs 18

5.3.2 Network Datagrams 18

5.3.3 Streaming Data 19

5.4 Constraints 20

5.5 Data Receiver Support 20

6. ATSC DATA BROADCAST ENCAPSULATION PROTOCOLS 20

6.1 ATSC Data Broadcast Encapsulation Protocols 20

6.1.1 Data Download Protocol 20

6.1.1.1 Overview 20

6.1.2 Identification and Versioning 22

6.1.3 Asynchronous Data Modules in the Data Download Protocol 23

6.1.4 Asynchronous Data Streaming in the Data Download Protocol 25

6.1.5 Non-Streaming Synchronized Data Modules (Synchronized Data Download Protocol) 25

6.1.6 Data Download Protocol Hierarchy 25

6.1.7 Data Download Protocol in an MPEG-2 Transport Stream Packet 26

6.1.8 DSMCC_section 27

6.1.9 DownloadServerInitiate (DSI) 28

6.1.10 DownloadInfoIndication (DII) 30

6.1.10.1 DII and DSI Descriptors 32

6.1.10.1.1 Module Link Descriptor 33

6.1.10.1.2 CRC32 Descriptor 33

6.1.10.1.3 Group Link Descriptor 34

6.1.11 DownloadDataBlock (DDB) 34

6.1.12 DownloadCancel (DC) 37

6.1.13 PSIP Announcement of the Data Download Protocol 38

6.1.13.1 Data Service Descriptor in PSIP 39

6.1.13.2 PID Count Descriptor 39

6.1.14 Discovery of a Data Download Protocol Program Element 39
6.1.14.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT) 39
6.1.15 Binding of a Data Download Protocol Program Element 40
6.1.15.1 Protocol Encapsulation 40
6.1.15.2 Download Descriptor 40
6.1.16 DSMCC_section CRC32 / Checksum Calculation 41
6.1.16.1 CRC32 Calculation 41
6.1.16.2 Checksum Encoding and Decoding 41
6.2 The ATSC DSM-CC Addressable Section 42
6.2.1 Overview 42
6.2.2 LLC Encapsulation 43
6.2.3 Datagram Transport 44
6.2.4 DSM-CC Addressable Section Stream Syntax 45
6.2.5 Data Transport Specification Usage 46
6.2.6 DSM-CC Addressable Section Mapping into MPEG-2 Transport Stream Packets 48
6.2.7 PSIP Announcement of the DSM-CC Addressable Section 49
6.2.7.1 Data Service Descriptor in PSIP 49
6.2.7.2 PID Count Descriptor 49
6.2.8 Discovery of a DSM-CC Addressable Section Program Element 50
6.2.8.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT) 50
6.2.9 Binding of a DSM-CC Addressable Section Program Element 50
6.2.9.1 Protocol Encapsulation 50
6.2.9.2 Multiprotocol Encapsulation Descriptor 51
6.3 Synchronous and Synchronized Data Streaming 51
6.3.1 Definitions 52
6.3.2 Encapsulation of PES Packets in Transport Stream Packets 52
6.3.3 Synchronous Data Streaming 53
6.3.4 Synchronous Data Streaming of Network Datagrams 58
6.3.5 Synchronized Data Streaming 59
6.3.6 Synchronized Data Streaming of Network Datagrams 63
6.3.7 PSIP Announcement of Synchronized and Synchronous Data Streams 64
6.3.7.1 Data Service Descriptor in PSIP 65
6.3.7.2 PID Count Descriptor 65
6.3.8 Discovery of Synchronous and Synchronized Data Streaming Program Elements 65
6.3.8.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT) 65
6.3.9 Binding of the Synchronized and Synchronous Data Streaming Program Elements 66
6.4 Data Piping 66
6.4.1 PSIP Announcement of the Data Piping Protocol 66
6.4.1.1 Data Service Descriptor in PSIP 67
6.4.1.2 PID Count Descriptor 67
6.4.2 Discovery of a Data Piping Protocol Program Element 67
6.4.2.1 PSIP Service Location Descriptor (SLD) and Program Map Table (PMT) 67
6.4.3 Binding of a Data Piping Protocol Program Element 68
6.4.3.1 Protocol Encapsulation 68

7. PSIP DATA SERVICE EVENT ANNOUNCEMENT 68
7.1 Data Service Descriptor 70
7.2 PID Count Descriptor 71
8. DATA SERVICE PROGRAM ELEMENT DISCOVERY AND BINDING 71
8.1 Association Tag Descriptor 76
8.2 Data Service Discovery–The Basic Algorithm 76
8.2.1 PSIP Structures 76
8.2.2 MPEG-2 Structures 77
8.2.3 A/90 Service Description Framework Information 77
8.3 Data Service Table (DST) 77
8.3.1 Tap Structures 81
8.4 Network Resource Table (NRT) 83
8.5 Service Discovery and Binding—Putting It All Together 85
9. BUFFER MODEL 87
9.1 Buffer Model for Asynchronous Data Elementary Streams 88
9.2 Smoothing Buffer Descriptor 89
9.3 Maximum Bitrate Descriptor 90
9.4 Buffer Model for Synchronous Data Elementary Streams 90
9.5 Buffer Model For Synchronized Data Services 91
9.5.1 T-STD for Synchronized Program Elements 91
9.5.2 Data Service Levels 94
9.5.2.1 Synchronized Stream Calculation Pseudo Code 95

Annex A: A Descriptor Location Matrix

1. SERVICE DESCRIPTION FRAMEWORK (SDF) DESCRIPTOR LOCATIONS 96
1.1 Data Download Protocol Descriptor Location Matrix 96

Annex B: ATSC Data Broadcast Encapsulation Protocols

1. SCOPE 97
1.1 Data Download Protocol Messages 97
1.1.1 DSM-CC Section 97
1.1.1.1 DSM-CC Message Header and DSM-CC Download Data Header 98
1.1.1.2 Download Server Initiate Message 100
1.1.1.3 Download Info Indication Message 102
1.1.1.4 Download Cancel Message 103
1.1.1.5 Download Data Block Message 104
1.1.1.6 Encapsulating Download Messages into MPEG-2 Transport Stream Packets 105

Annex C: Sample Encapsulations

1. DATA DOWNLOAD PROTOCOL EXAMPLE 107
1.1 Sample Tables 107

1.1.1 Sample DSI 107
1.1.2 Sample DII for Group 1 (English) 109
1.1.3 Sample DDB (for English module) 111
1.1.4 Sample DII for Group 1 (French) 113
1.1.5 Sample DDB (for French module) 115
1.2 DSM-CC Addressable Section Example 116
1.3 Data Piping Example 118

Annex D: ATSC Data Broadcast Encapsulation Protocols

1. SCOPE 120
1.1 ATSC Table Hierarchy Discussion 121

Index of Tables and Figures

Table 5.1 Encapsulation Protocol Selection Matrix 17
Table 6.1 TransactionId Subfields 23
Table 6.2 MPEG-2 Transport Stream Packet Header for DSM-CC Section 27
Table 6.3 DSI Message 28
Table 6.4 DII Message 30
Table 6.5 II and DSI Descriptors 33
Table 6.6 Module Link Descriptor Syntax 33
Table 6.7 CRC32 Descriptor Syntax 34
Table 6.8 Group Link Descriptor Syntax 34
Table 6.9 DDB Message 35
Table 6.10 DC Message 37
Table 6.11 Download Descriptor Syntax 41
Table 6.12 DSMCC_addressable_section Syntax 46
Table 6.13 MPEG-2 Transport Packet Header for DSM-CC addressable section 49
Table 6.14 Multiprotocol Encapsulation Descriptor Syntax 51
Table 6.15 Synchronous Data in PES 54
Table 6.17 Synchronized Streaming Data in PES 60
Table 6.18 Synchronous and Synchronized Protocol_Encapsulation Field Values 66
Table 7.1 Data Service Descriptor Syntax 71
Table 7.2 PID Count Descriptor Syntax 71
Table 8.1 Association Tag Descriptor 76
Table 8.2 Data Services Table 78
Table 8.3 Network Resources Table 84
Table 9.1 Smoothing Buffer Descriptor Location Matrix 90
Table 9.2 Maximum Bitrate Descriptor Location Matrix 90
Table A1 SDF Descriptor Location Matrix 96
Table A2 Data Download Protocol Descriptor Location Matrix 96
Table B1 DSM-CC Section Format 98
Table B2 DSM-CC Message Header 99
Table B3 DSM-CC Download Data Header 99
Table B4 DSM-CC Adaptation Header 100
Table B5 Download Server Initiate Message 101
Table B6 Group Link Descriptor 102
Table B7 Download Info Indication Message 102
Table B8 Module Link Descriptor 103
Table B9 CRC32 Descriptor 103
Table B10 Download Cancel Message 104
Table B11 Download Data Block Message 104
Table B12 MPEG-2 Transport Stream Packet Header 105
Table C1 Sample DSI 107
Table C2 Sample DII for Group 1 (English) 110
Table C3 Sample DDB (for English module) 111
Table C4 Sample DII for Group 1 (French) 113

Table C5 Sample DDB (for French module) 115
Table C6 DSM-CC Addressable Section Example 117
Table C7 Data Piping Example 119

Figure 5.1 Graphical encapsulation overview and relation to other standards. 13
Figure 5.2 Data service components. 14
Figure 5.3 ATSC Data Broadcast protocol packetization, synchronization, and protection layers. 15
Figure 6.1 Structure of the one-layer and two-layer download scenarios. 22
Figure 6.2 Cyclic transmission of information in a data carousel. 24
Figure 6.3 Data download protocol module fragmentation. 26
Figure 6.4 LLC encapsulation. 43
Figure 6.5 SNAP header. 43
Figure 6.6 LLC/SNAP encapsulation. 44
Figure 6.7 Device ID mapping. 45
Figure 6.8 DSM-CC addressable section syntax. 46
Figure 6.9 A DSMCC_addressable_section encapsulated and mapped into one or more MPEG-2 Transport Stream packets. 48
Figure 6.10 Encapsulation of synchronous PES packets in MPEG-2 Transport Stream packets. 53
Figure 6.11 Encapsulation of IP packets for synchronous data streaming using PES frames. 59
Figure 6.12 Encapsulation of IP packets for synchronized data streaming using PES frames. 64
Figure 7.1 Event Table hierarchy. 69
Figure 8.1 The Relationship of the VCT, PMT, DST, and NRT in the SDF 75
Figure 8.2 Service description block diagram.. 86
Figure 9.1 Transport system target data decoder model for asynchronous data elementary streams. 88
Figure 9.2 System target data decoder buffer model for synchronized data services. 92
Figure B1 Packing DSM-CC Sections into MPEG-2 Transport Stream Packets 106
Figure D1 Example ATSC table hierarchy. 120

Significant Contributors to this Document

John R. Mick Jr., SkyStream Networks Corporation
Rich Chernock, IBM Corporation
Regis Crinon, Sharp Laboratories of America/Intel Corporation
Edwin Heredia, Samsung Information Systems America/Microsoft Corporation
Art Allison, National Association of Broadcasters
Michael A. Dolan, DIRECTV

Recommended Practice:

Implementation Guidelines for the ATSC Data Broadcast Standard

1. SCOPE

This document provides a set of guidelines for the use and implementation of the Advanced Television Systems Committee (ATSC) Data Broadcast Standard. These guidelines are intended to be recommendations for the usage of the ATSC Data Broadcast Standard as described in the ATSC Standard, A/90 (2000), "Data Broadcast Standard" [1]. As such, they facilitate the efficient and reliable implementation of data broadcast services. The information contained herein applies to data service providers as the primary entity that assembles the elements of each data channel. It also applies to broadcasters, network operators, and infrastructure 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 document uses the terminology defined in ATSC Data Broadcast Standard [1] and should be read in conjunction with [1].

2. REFERENCES

[1] ATSC Standard A/90 (2000), " ATSC Data Broadcast Standard"

[2] ISO/IEC 13818-1: "Information technology—Generic coding of moving pictures and associated audio information, Part 1: Systems—International Standard (IS)"

[3] ISO/IEC 13818-2: "Information technology—Generic coding of moving pictures and associated audio information, Part 2: Video—International Standard (IS)"

[4] ISO/IEC 13818-6: "Information technology—Generic coding of moving pictures and associated audio information, Part 6: Extension for Digital Storage Media Command and Control (DSM-CC), International Standard (IS)"

[5] IETF RFC 791: "Internet Protocol", J. Postel, 01.09.1981

[6] ATSC Standard A/65A (2000), "Program and System Information Protocol (PSIP) for Terrestrial Broadcast and Cable"

[7] IETF RFC 1112: "Host Extensions for IP Multicasting"

[8] ISO/IEC 8802-2 (ANSI/IEEE standard 802.2), "Information technology—Telecommunications and information exchange between systems, Local and metropolitan area networks, Specific requirements, Part 2: Logical link control"

[9] IETF RFC1042: "A standard for the Transmission of IP Datagrams over IEEE 802 Networks," J. Postel and J. Reynolds, February 1988.

3. ACRONYMS AND DEFINITIONS

3.1 Acronyms

The following acronyms are used within this document:

ATSC	Advanced Television Systems Committee
ATVEF	Advanced Television Enhancement Forum
CRC	Cyclic Redundancy Check
DAU	Data Access Unit
DAVIC	Digital Auido Visual Council
DC	DownloadCancel
DDB	DownloadDataBlock
DEBn	Data Elementary Stream Buffer
DET	Data Event Table
DII	DownloadInfoIndication
DSI	DownloadServerInitiate
DST	Data Service Table
DSM-CC	Digital Storage Media Command and Control
DTV	Digital Television
EIT	Event Information Table
EPG	Electronic Program Guide
ES	Elementary Stream
ESCR	Elementary Stream Clock Reference
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IPX	Internetwork Packet Exchange
LLC/SNAP	Logical Link Control and Subnetwork Attachment Point
LSB	Least Significant Byte
LTST	Long Term Service Table
MGT	Master Guide Table
MPEG	Moving Picture Experts Group
MTU	Maximum Transmission Unit

NRT	Network Resources Table
PAT	Program Association Table
PCR	Program Clock Reference
PES	Packetized Elementary Stream
PID	Packet Identification
PMT	Program Map Table
PSIP	Program and System Information Protocol
PTS	Presentation Time Stamp
RFC	Request For Comment
SBn	Smoothing Buffer
SDF	Service Description Framework
SLD	Service Location Descriptor
TBn	Transport Buffer
TS	Transport Stream
U-N	User to Network
VCT	Virtual Channel Table

3.2 Definitions

For definitions used in this document, see the ATSC Data Broadcast Standard [1].

3.3 This Document's Table Structure Format

The syntactic tables located in this document have been constructed such that the data structure definitions are centralized within the description. Thus, a table element having sub-elements has been described sequentially within the table so that implementers can easily reference all the data elements of the data structure. The typical syntax lists the name of the "major" element and then indents for all elements and sub-elements of the data structure.

Binary field values in tables are sometimes designated by single quotes, such as '0' or '01', and sometimes by a 'b' suffix, such as 0b or 01b.

4. INTRODUCTION TO THE IMPLEMENTATION GUIDELINES OF THE ATSC A/90 DATA BROADCAST SPECIFICATION

This document contains recommendations on the usage of the Advanced Television Systems Committee (ATSC) Data Broadcast Standard [1]. The Implementation Guide examines in detail many of the components of the ATSC Data Broadcast Standard [1]. The Implementation Guide is organized as follows:

- Section 5 provides a high-level general overview of the ATSC Data Broadcast Standard [1] including criteria for selecting the appropriate encapsulation protocol.

- Section 6 explores in detail each encapsulation protocol including the examination of each field of the encapsulation protocol. Additionally, suggestions regarding the usage of the fields are provided, along with the field ranges and other valuable information.
- Section 7 focuses on the announcement of a Data Service in the ATSC environment using the Program and System Information Protocol (PSIP).
- Section 8 examines the Service Description Framework (SDF) used by applications to discover their resources and to bind the resources to the applications.
- Section 9 focuses on the buffer models specified by the ATSC Data Broadcast Standard [1].

Additionally, the Implementation Guideline contains four annexes. The annexes cover the following:

- Annex A contains a matrix of the descriptors used by the ATSC Data Broadcast Standard [1] and the table locations in which they may be used.
- Annex B describes the messages that are used for the data broadcast encapsulation protocols defined in the ATSC Data Broadcast Standard [1].
- Annex C provides examples showing the encoding of a simple two-layer asynchronous non-streaming Data Service via the non-flow controlled download scenario.
- Annex D provides an example of the Service Description Framework's (SDF) construction of a Data Service that contains the Program elements of Annex C. The SDF example illustrates the Program elements announcement using PSIP and the Program elements discovery along with the application's binding to the Program elements.

The Implementation Guide should be read in conjunction with the ATSC Data Broadcast Standard [1]. In the case of any discrepancies between the two documents then the ATSC Data Broadcast Standard [1] takes precedence.

5. INTRODUCTION TO THE ATSC A/90 DATA BROADCAST STANDARD

The ATSC A/90 Data Broadcast Standard [1] describes the available encapsulation protocols used to transport data within the ATSC MPEG-2 Transport multiplex. The specification explains the Service Description Framework (SDF) used for the discovery of Program elements and the binding of applications to their Program Elements. Additionally, the specification describes the additions to the ATSC Program and System Information Protocol (PSIP) standard [6] such that a Data Service may be announced within the existing Electronic Program Guide (EPG) mechanisms. The specification also explains the use of elements in MPEG-2 PSI to aide the SDF for binding of applications to their Program Elements. It must also be noted that the ATSC Program and System Information (PSIP) standard [6] and MPEG-2 PSI (PAT and PMT) aid the SDF with the discovery of Program Elements. Figure 5.1 provides an overview of the ATSC Data Broadcast Standard as described in [1].

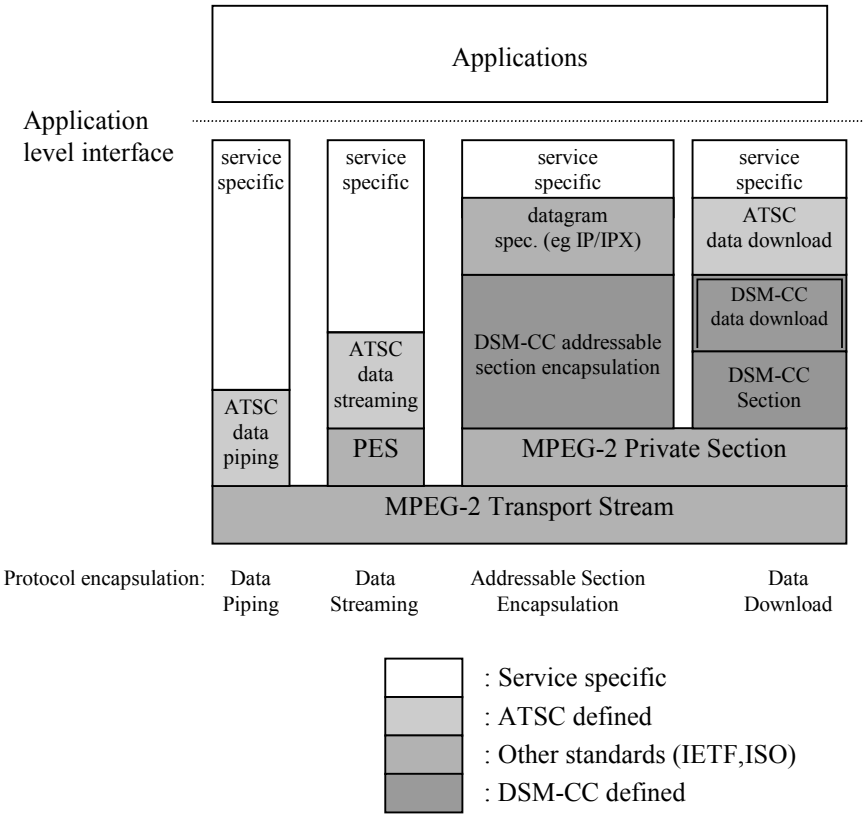


Figure 5.1 Graphical encapsulation overview and relation to other standards.

The basis of the ATSC Data Broadcast Standard [1] is formed by the MPEG-2 Transport Stream (TS) as defined in ISO/IEC 13818-1 (MPEG-2 systems) [2] and amendments 1 and 2 specifying registration procedures for the copyright identifier and the format identifier, respectively. Data information can be transported within this MPEG-2 TS by means of the following encapsulation protocols:

- Data Piping
- Data Streaming
- Addressable Sections
- Data Download

Figure 5.1 identifies what is standardized and by which body. ISO has standardized the MPEG-2 TS in ISO/IEC 13818-1 [2] and the DSM-CC framework in ISO/IEC 13818-6 [4]. The IETF has standardized the Internet Protocol (IP) in RFC 791 [5]. The ATSC has specified within the ATSC Data Broadcast Standard [1] the ATSC Data Piping protocol, the ATSC data streaming encapsulation, the ATSC DSM-CC addressable section encapsulation, and the ATSC Download protocol. Within Figure 5.1, the encapsulation of the Internet Protocol (IP) is just an example. Other network protocols can also be encapsulated.

As shown in Figure 5.1, the ATSC Data Broadcast Standard [1] specifies different encapsulation protocols for different application areas.

The Data Piping specification provides minimal information on how to acquire and assemble the data bytes from the MPEG-2 TS. It only specifies how to put data into the MPEG-2 Transport Stream packets.

The data streaming specification provides additional functionality, especially for timing. It is possible to achieve synchronous data broadcast or synchronized data broadcast (see Section 5.3). The data streaming specification is based on PES packets as defined in MPEG-2 ISO/IEC 13818-1 [2].

The ATSC DSM-CC addressable section encapsulation and the ATSC Download protocol specifications are built using the DSM-CC framework of MPEG-2 ISO/IEC 13818-6 [4]. The mapping of the DSM-CC protocols onto MPEG-2 TS utilizes the MPEG-2 Private Sections as defined in MPEG-2 ISO/IEC 13818-1 [2].

The ATSC Data Broadcast Standard [1] has added specific information in order for the framework to work within the ATSC environment, especially in conjunction with the Program and System Information Protocol (PSIP) [6].

In the ATSC Data Broadcast standard, each data service may be composed of one or more applications integrated to the remaining ATSC infrastructure by means of Announcement, Discovery, and Binding functions (See Figure 5.2).

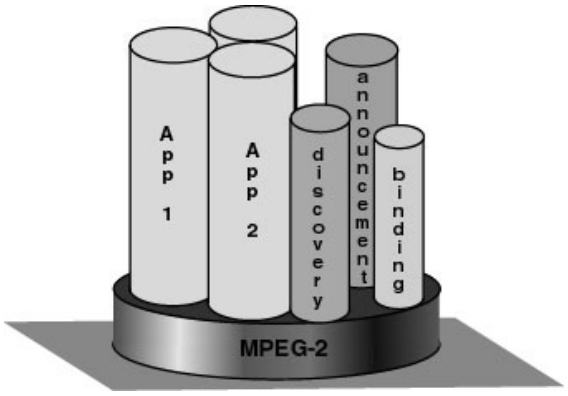


Figure 5.2 Data service components.

The announcement and discovery specification is part of the ATSC A/65 Program and System Information Protocol (PSIP) specification [6], along with the new elements documented in the ATSC Data Broadcast Standard [1]. Additionally, further discovery definition and application binding are part of the Service Description Framework (SDF) that is described in the ATSC Data Broadcast Standard [1]. Finally, data delivery is part of the ATSC Data Broadcast Standard [1].

Figure 5.3 illustrates the packetization, synchronization, and protection layers of the ATSC Data Broadcast protocols as per [1].

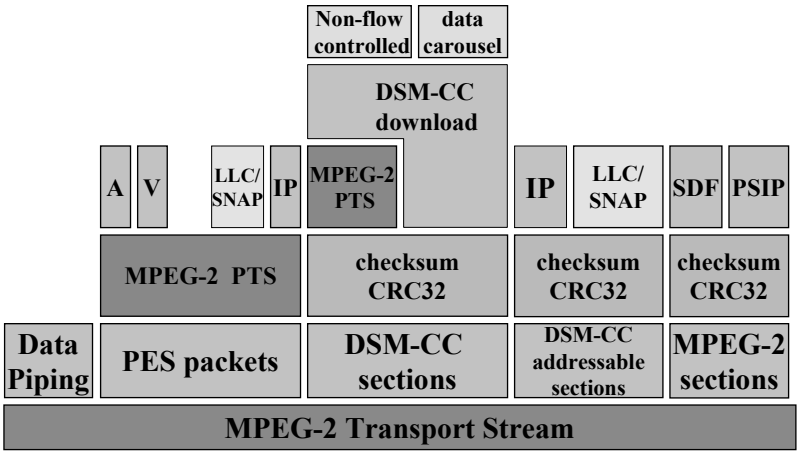


Figure 5.3 ATSC Data Broadcast protocol packetization, synchronization, and protection layers.

The following paragraphs provide a set of implementation guidelines regarding how to utilize the different encapsulation protocols.

5.1 Data Service Definition

A Data Service is the collection of applications and associated resources signaled in the Data Service Table (DST) of the Service Description Framework. A Data Service is required to have one program element (data elementary stream) conveying the DST and optionally the Network Resources Table (NRT). A virtual channel, as described by the PSIP Virtual Channel Table (VCT), may include a maximum of a single Data Service. The minor_channel_number of a data-only channel (service_type 0x04 in the VCT) must be in the range 100-999. A Data Service may be composed of any number of program elements and other resources, and the program elements may include any combination of the encapsulation protocols specified in the ATSC Data Broadcast Standard.

The discovery of a Data Service in an ATSC Transport multiplex is independent of the announcement of a Data Service. The announcement procedure formalizes the technique used to signal the Data Service event (i.e., the start time and the duration) during which the Data Service is active. The discovery procedure formalizes the mechanism used for identifying the presence of a Data Service and for identifying the components comprising the Data Service. The binding mechanism describes a way to provide application specific parameters and resource signaling. The announcement of a Data Service is documented in Section 7 herein and Section 11 of the ATSC Data Broadcast Standard [1]. The discovery and binding of a Data Service is discussed in Section 8 of this document and Section 12 of the ATSC Data Broadcast Standard [1].

5.2 Data Encapsulations

As illustrated in Figure 5.1 and Figure 5.3, there are multiple ways to encapsulate data within the ATSC Transport multiplex. The mechanisms have different characteristics concerning timing,

filtering, overhead, size, etc. The selection of the appropriate mechanism is based on the specific requirements of the target application.

The level of detail of the ATSC Data Broadcast Standard [1] varies for the different encapsulation protocols. In the case of the DSM-CC addressable section encapsulation (see Section 6.2 of this document and Section 8 of the ATSC Data Broadcast Standard [1]) and the Download protocol (see Section 6.1 of this document and Section 7 of the ATSC Data Broadcast Standard [1]), the specification is very detailed. Thus, this Implementation Guideline only requires very few application specific definitions. In the case of the other protocols, there is significant freedom for implementers to make their own decisions.

5.2.1 Packetization of Data Entities

Generally, data of any protocol is transmitted in a packetized form ("data entities"). These data entities may have different lengths. If the data is not packetized or the packetization method is irrelevant or hidden to the ATSC transmission chain the most appropriate way of transmission is Data Piping (see Section 6.4 of this document and Section 10 of the ATSC Data Broadcast Standard [1]).

At the MPEG-2 Transport Stream layer, data is transmitted within MPEG-2 Transport Stream packets with a fixed length of 188 bytes (184 bytes payload); therefore, data entities of higher layers must often be split at the transmission side and must be re-assembled at the reception device. For the splitting of the data entities, there are several possible packetization mechanisms:

- Private mechanisms based on the Data Piping
- MPEG-2 Packetized Elementary Streams (PES)
- MPEG-2 Private Sections
- DSM-CC Data Download Blocks

MPEG-2 PES provides a mechanism to transmit data entities 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-2 for synchronization of Video and Audio). MPEG-2 PES was chosen by ATSC Data Broadcast Standard [1] for the transmission of all synchronous data streams and for synchronized data streaming (see Section 6.3 in this document and Section 9 of the ATSC Data Broadcast Standard [1]).

MPEG-2 Private Sections can be used to transmit data entities of variable size with a maximum length for each data entity of just less than 4 Kbytes (encapsulation specific). The transmission is asynchronous. Data entities can be recombined to form larger data objects (for example, modules reassembled from download data blocks). MPEG-2 Private Sections are built in a way that MPEG-2 demultiplexers currently available on the market can filter out single sections in hardware, thereby reducing the required software processing power of the receiver. This concept is the main reason why the MPEG-2 Private Sections have been chosen as the mechanism for the transmission of asynchronous network protocols and data downloads (Download protocol). The Download protocol provides a means of re-assembling sections (DownloadDataBlocks) into modules, where the section data may be up to 4066 bytes in size. DSM-CC Addressable sections are used for the carriage of data entities like IP datagrams. Again, these sections are designed to engage the section filtering capability of receivers.

5.3 Encapsulation Protocol Selection Guidance

The following text offers guidance for when a particular encapsulation technique might be selected. The list is not all encompassing; it is provided as a starting point for the selection of an encapsulation mechanism.

Table 5.1 provides a generalized guide for determining the recommended encapsulation method. The column headings represent the data size characteristics. The row headings represent the data timing or synchronization requirements.

Table 5.1 Encapsulation Protocol Selection Matrix

	Bounded	**Unbounded/Streaming**	**Network Datagram**
Asynchronous	Asynchronous Module Download Protocol	Asynchronous Data Streaming Download Protocol	DSM-CC Addressable Sections
Synchronous		Synchronous Data Streaming (PES)	Synchronous Data Streaming (PES)
Synchronized	Synchronized Data Download Protocol	Synchronized Data Streaming (PES)	Synchronized Data Streaming (PES)

The delivery of asynchronous data elementary streams is not subject to any timing constraints derived from the presence of MPEG-2 Systems timestamps in the ATSC Transport Stream. In essence, asynchronous data elementary streams are not time sensitive streams and the bytes of such streams are delivered at a fixed leak rate from a well-defined Transport System Target Decoder (T-STD) buffer model to the data receiver. Asynchronous data elementary streams have stream_type value 0x0B, 0x0D or 0x95.

Synchronous data elementary streams are used for applications requiring continuous streaming of data to the receiver at a regular and piece-wise constant data rate. Synchronous data is delivered as a stream of 2-byte data access units, each data access unit being associated with a precise delivery time derived directly from the PTS field in the PES packet header and a leak rate specified either in the PES packet header or the synchronous data header structure at the beginning of the PES packet payload. The Presentation Time Stamps in the PES packets specifies the time of delivery of the first synchronous data access unit in the PES packet relative to the System Time Clock (STC) reconstructed from the PCR fields in the ATSC Transport Stream. The leak rate is then used to infer the delivery time for each of the following synchronous access units within the same PES packet. Delivery of the synchronous data access units in the next PES packet starts at the time specified by the PTS of that next PES packet. The PCR timestamps may be delivered in the Transport Packets conveying the synchronous data elementary stream. The delivery of synchronous data access units is governed by a well-defined T-STD buffer model. The difference between synchronous data and asynchronous data is the fact that synchronous data access units have been defined for the sole purpose of delivering bytes out of the T-STD buffer model according to a strict timing tied to the 27 MHz System Time Clock of the receiver. Synchronous data elementary streams have stream_type value 0xC2.

Synchronized data elementary streams are used for applications requiring presentation of data at precise but not necessarily regular instants. The presentation times are typically, but not necessarily, associated with some other reference video, audio or data elementary stream. Synchronized data is delivered as a series of Data Access Units spanning one or several PES packets. The time separating two consecutive synchronized Data Access Units is arbitrary. A synchronized data elementary stream includes Presentation Time Stamps. The Program Clock Reference timestamps are typically delivered in the MPEG-2 Transport Stream packets of a

reference elementary stream, but they may be in the same elementary stream as the synchronized data. The purpose of the PTS timestamps is to specify the instant in time, relative to the STC, at which a data access unit must be rendered/displayed in the receiver. The T-STD for synchronized data elementary streams includes an additional buffer for re-assembling data bytes into synchronized Decoding Access Units. Therefore, as opposed to synchronous data access units, synchronized data access units have been defined for the purpose of presenting data at precise times. Also, as opposed to synchronous data access units, consecutive synchronized data access units may not be of the same size. Consequently, the T-STD for synchronized data elementary streams goes one step further by providing a buffer for collecting data bytes pertaining to a Data Access Unit before it is decoded and presented at a precise instant of the receiver System Time Clock. Synchronized data elementary streams have stream_type value 0x06 or 0x14.

5.3.1 Bounded Data Blobs

A data blob or a data module could be a representation of a file, a group of files, a directory of files, a group of directories containing files, a module, a hierarchy of modules, an object, a group of objects, or a bounded data entity whose boundaries are known to the content author. The definition of a data module is not defined by this document or by the ATSC Data Broadcast Standard [1]. However, for the purpose of this section, a data module can generally be viewed to represent a finite sized entity that is to be delivered to a data receiver. Data modules that have no synchronization requirements associated with them should be delivered using the asynchronous module delivery method of the Download protocol. (See Section 6.1.3 of this document and Section 7 of the ATSC Data Broadcast Standard [1] for additional information.)

Data modules that are to be repetitively delivered should use the data carousel option of the asynchronous Download protocol. Otherwise, the non-flow controlled scenario of the Download protocol should be used.

Synchronized data modules should be delivered using the synchronized Download protocol. (See Section 6.1 of this document and Section 7 of the ATSC Data Broadcast Standard [1] for additional information.)

5.3.2 Network Datagrams

Network datagrams, for example Internet Protocol (IP) datagrams or Internetwork Packet Exchange (IPX) datagrams, can be encapsulated in various manners. The different encapsulation methods are intended to meet specific needs of different applications.

ATSC DSM-CC addressable sections are the method of delivery for asynchronous network datagrams. Datagrams that do not have any timing or synchronization requirements associated with them should be delivered using the DSM-CC addressable sections encapsulation. (See Section 6.2 in this document and Section 8 of the ATSC Data Broadcasting Standard [1] for additional information.) For example, this method of delivery would be useful for delivering web pages, providing turbo-internet service, or for delivering Advanced Television Enhancement Forum (ATVEF) content.

Datagrams that are synchronized to a program by using a Presentation Time Stamp (PTS) are conveyed in the synchronized streaming protocol. (See Section 6.3.5 in this document and Section 9 of the ATSC Data Broadcasting Specification [1] for additional information Examples of synchronized datagrams include datagram triggers, and audio/video associated enhanced data.

Synchronous datagrams are carried using the synchronous streaming protocol.

5.3.3 Streaming Data

Streaming data can be synchronous, synchronized, or asynchronous. Asynchronous streaming data should be carried by the asynchronous data Download protocol. (See Section 6.1.4 in this document and Section 7 of the ATSC Data Broadcast Standard [1] for additional information.)

The delivery of asynchronous data elementary streams is not subject to any timing constraints derived from the presence of MPEG-2 Systems timestamps in the ATSC Transport Stream. In essence, asynchronous data elementary streams are not time sensitive streams and the bytes of such streams are delivered at a fixed leak rate from a well-defined Transport System Target Decoder (T-STD) buffer model to the data receiver. Asynchronous data elementary streams have stream_type value 0x0B, 0x0D or 0x95.

All data that is synchronous is carried using the synchronous streaming encapsulation where the data is transported in Packetized Elementary Stream (PES) packets. The synchronous data streaming protocol has been harmonized with SCTE DVS132. (See Section 6.3.3 in this document and Section 9 of the ATSC Data Broadcast Standard [1] for additional information.) Synchronous data elementary streams are used for applications requiring continuous streaming of data to the receiver at a regular and piece-wise constant data rate. Synchronous data is delivered as a stream of 2-byte synchronous_data_access_units, each synchronous_data_access_unit being associated with a precise delivery time derived directly from the PTS field in the PES packet header and a leak rate specified either in the PES packet header or the synchronous data header structure at the beginning of the PES packet payload. The Presentation Time Stamps in the PES packets specifies the time of delivery of the first synchronous_data_access_unit in the PES packet relative to the System Time Clock (STC) reconstructed from the PCR fields in the ATSC Transport Stream. The leak rate is then used to infer the delivery time for each of the following synchronous_data_access_units. The PCR timestamps may be delivered in the Transport Packets conveying the synchronous data elementary stream. The delivery of synchronous_data_access_units is governed by a well-defined T-STD buffer model. The difference between synchronous data and asynchronous data is the fact that synchronous_data_access_units have been defined for the sole purpose of delivering bytes out of the T-STD buffer model according to a strict timing tied to the 27 MHz System Time Clock of the receiver. Synchronous data elementary streams have stream_type value 0xC2.

Synchronized data should be carried in the synchronized streaming protocol. The synchronized streaming protocol has been harmonized with DVB. (See Sections 6.3.5 and 6.1.5 respectively, and Sections 9 and 7 respectively of the ATSC Data Broadcast Standard [1] for additional information.) Synchronized data elementary streams are used for applications requiring presentation of data at precise but not necessarily regular instants and in connection to a reference video, an audio or another data elementary stream. Synchronized data is delivered as a series of Data Access Units spanning one or several PES packets. The time separating two consecutive synchronized Data Access Units is arbitrary. A synchronized data elementary stream includes Presentation Time Stamps. The Program Clock Reference timestamps are delivered in the MPEG-2 Transport Stream packets of the reference elementary stream. The purpose of the PTS timestamps is to specify the instant in time, relative to the STC, at which a data access unit must be rendered/displayed in the receiver. The T-STD for synchronized data elementary streams includes an additional buffer for re-assembling data bytes into synchronized Decoding Access Units. Therefore, as opposed to synchronous data access units, synchronized data access units have been defined for the purpose of presenting data concurrently with another media stream. Consequently, the T-STD for synchronized data elementary streams goes one step further by providing a buffer for collecting data bytes pertaining to a Data Access Unit before it is

decoded and presented at a precise instant of the receiver System Time Clock. Synchronized data elementary streams have stream_type value 0x06 or 0x14.

5.4 Constraints

Section 1.3 of the ATSC Data Broadcast Specification A/90 [1] constrains the data broadcast standard in the following fashion:

"This standard shall not be used for the carriage of the following elementary streams:

- Video elementary streams with stream_type 0x02
- Audio elementary streams with stream_type 0x81

Other ATSC standards define transmission of elementary streams of these two stream types."

The purpose of these constraints is to prevent the use of A/90 to transmit program video and audio in non-ATSC authorized forms and was not intended to prevent non-programmatic video and audio from being transmitted as data.

5.5 Data Receiver Support

A data receiver may support any combination of the encapsulation protocols. A data receiver that encounters an unsupported encapsulation protocol, or any other unsupported data item delivered via the Transport multiplex, should seamlessly discard the unsupported information.

6. ATSC DATA BROADCAST ENCAPSULATION PROTOCOLS

The following sections describe the ATSC Data Broadcast Standard [1] encapsulation protocols described in Sections 7, 8, 9, and 10 of [1].

6.1 ATSC Data Broadcast Encapsulation Protocols

The following sections describe the ATSC Data Broadcast Standard [1] encapsulation protocols described in Sections 7, 8, 9, and 10 of [1].

6.1.1 Data Download Protocol

The data download protocol is specified in Section 7 of the ATSC Data Broadcast Standard [1].

6.1.1.1 Overview

The ATSC Data Broadcast Standard uses the DSM-CC User-To-Network Download Protocol, as defined in 13818-6 [4], to carry four different types of data:

- Carousel delivery (cyclic transmission) of finite data modules to a receiver
- One time, non-flow controlled delivery of finite data modules to a receiver
- Non-flow controlled, asynchronous delivery of streaming data to a receiver
- Non-flow controlled, synchronized delivery of non-streaming data to a receiver

This section provides an introduction to the general features of the data download protocol that are common to all of these four types of data download scenario.

Within the following discussion, the term *module* is a generic term to describe an ordered collection of bytes. It should not be interpreted to imply a file or specific object boundary. The nature of the items the term represents is outside the scope of ATSC Data Broadcast Standard [1] and of this document.

The Download Protocol is used to deliver modules. Each module is broken up into *blocks* for the purpose of transmission. The blocks of a module must all be of the same size, except for the last block of the module, which may be smaller. Each block may have up to 4066 bytes of data for the data carousel scenario, one-time finite module scenario, or asynchronous streaming data scenario, and up to 4058 bytes for the synchronized non-streaming data scenario. The maximum number of blocks in a module is 65,535. Therefore, a module may contain up to 266,469,376 bytes in the first three scenarios and up to 265,945,088 bytes in the synchronized non-streaming data scenario.

The modules are clustered into *groups*, and the groups in turn may be clustered into *supergroups*. The supergroups are useful for two reasons:

- They can be used to provide a two level hierarchical structuring of the modules.
- They can be used to allow a larger number of modules in the download scenario than would be possible with the group structure alone.

A group may contain up to 507 modules. A supergroup may contain up to 405 groups. Therefore, a single download scenario may contain up to 205,335 modules.

There is a mechanism called the module_link_descriptor that can be used to link multiple modules from the same group into a single logical data item, so it is possible to transmit individual data items that are much larger than the maximum module size. There is also a mechanism called the group_link_descriptor that can be used to link any subset of groups within a supergroup into a single logical grouping.

The Download Protocol uses several types of *messages* to manage the delivery of the data:

- A DownloadDataBlock (DDB) message delivers an individual block of a module. It includes an identifier of the module and a 16-bit block number giving the position of the block in the module.
- A DownloadInfoIndication (DII) message describes the modules in a group. It uses a moduleInfoBytes field to identify the modules in the group and give the size of the blocks used to deliver the modules, the size and version number of each module, and possibly other information about each module.
- A DownloadServerInitiate (DSI) message describes the groups in a supergroup. It uses a groupInfoBytes field to identify the DII messages for the groups in the supergroup and give the size, version number, and possibly other information about each group.
- A DownloadCancel (DC) message indicates the termination of a download scenario; i.e., it indicates that no more DDB, DII, or DSI messages will be forthcoming. (This allows a receiver to reclaim all resources such as buffers, etc., allocated to the download scenario.)

All of the messages of a single download scenario must be in the same data elementary stream (i.e., be contained in transport packets having the same PID).

The DownloadDataBlock message is usually called a download data message, while the other three types of messages are usually called download control messages. Typically the DSI and DII messages will be transmitted repeatedly, even when they have not changed, to accommodate receivers that tune to the data service in the middle of a download scenario. Thus, a receiver will often receive multiple instances of the same version of these messages.

A scenario consisting of only a single group is called a one-layer download scenario. A scenario consisting of a supergroup containing multiple groups is called a two-layer download scenario. In the case of a one-layer scenario the DII message is the top-level control message. In the case of a two-layer scenario the DSI message is the top-level control message.

A Data Service provider may use a one-layer or two-layer scenario, depending on the total number of modules to be delivered and the perceived value of having a two-layer structuring for them. A data receiver should support both one-layer and two-layer download scenarios.

Figure 6.1 illustrates both one-layer and two-layer download scenarios.

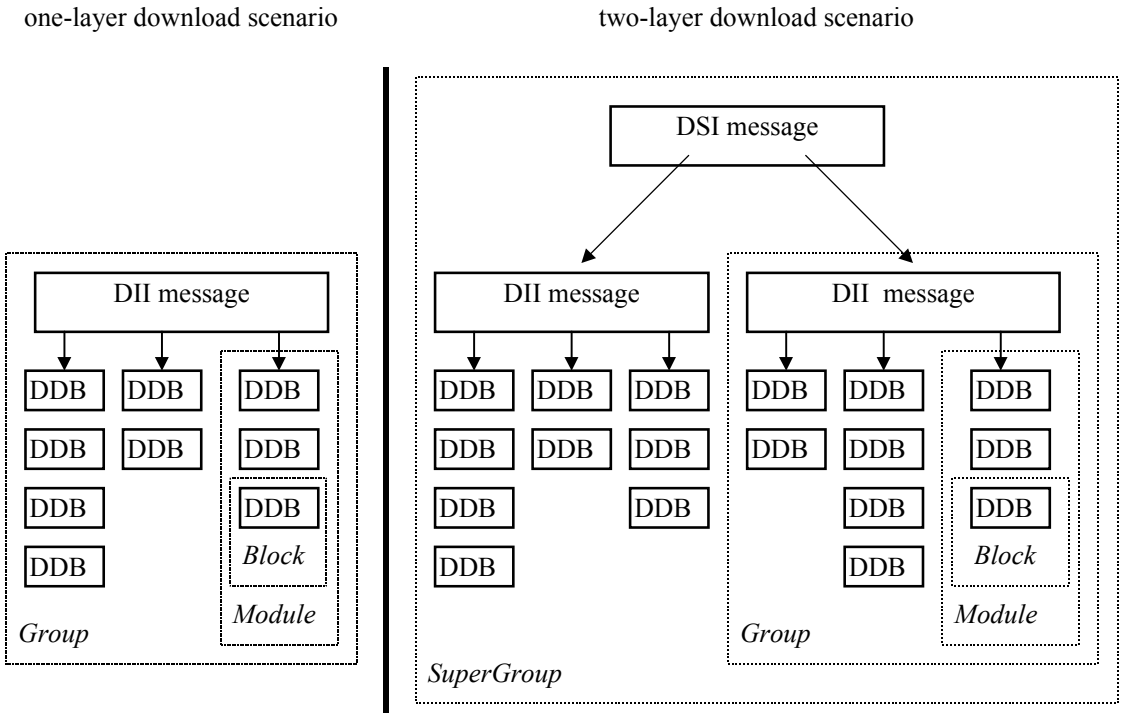


Figure 6.1 Structure of the one-layer and two-layer download scenarios.

Every module in the download scenario must be referenced by a DII message of the scenario. It is also strongly recommended that in a two-layer download scenario every group should be referenced by the DSI message of the scenario.

6.1.2 Identification and Versioning

Each download scenario has associated with it a 32-bit identifier called the downloadId, which appears as a field in the DII, DC, and DDB messages of the scenario. This identifier may be chosen arbitrarily, except that it must be unique within the transport stream.

Each module has associated with it a 16-bit identifier called the moduleId. This moduleId may be selected arbitrarily, except that it may not be in the range 0xFFF0 to 0xFFFF, and it must be unique within the download scenario to which the module belongs. This implies that groups can share modules, since the moduleId is not scoped by the group.

The DII messages use the moduleId to reference the modules in the group. Also, each DDB message uses the moduleId to identify the module to which the block in the DDB message belongs.

Each module has associated with it an 8-bit version number called the moduleVersion, which appears in the DDB messages for the blocks of the module, and in the DII messages pointing to the module. Neither the DSM-CC standard nor the ATSC Data Broadcast Standard specify how module versioning should be handled, except for an implicit assumption that the moduleVersion changes whenever the content of the module changes. The recommended practice is to increment the moduleVersion whenever the content of the module changes, with wrap-around from 255 to 0.

Each download control message (DSI, DII, or DC) has associated with it a 32-bit number called the transaction_id, which includes both a unique identifier and a version number for the message. The subfields of this transaction_id are defined as shown in Table 6.1, where bit 31 is the high order bit and bit 0 is the low order bit:

Table 6.1 TransactionId Subfields

Subfield	No. of Bits	Value
originator (transaction_id[31..30])	2	'10', indicating transaction_id is assigned by the server
version (transaction_id[29..16])	14	arbitrary, but must be modified whenever message is changed
identification (transaction_id[15..1])	15	0 for top-level control message (DSI message for two-layer scenario, DII message for one-layer scenario) non-zero value for other control messages (DII message for two-layer scenario, DC message in all cases), unique within the download scenario, serves as group identifier in two-layer download scenario.
updated_flag (transaction_id[0])	1	must be toggled every time the control message is updated

The recommended practice is to increment the value in the version subfield whenever the message is changed, with wraparound to 0 when the top of the range is reached, and to reflect the lowest order bit of this subfield in the updated_flag subfield.

Note that a receiver can always locate the top-level control message of a data download scenario by just looking for a control message with the identification subfield of the transaction_id equal to 0.

The DSI messages use the transaction_id of the DII messages to reference the groups of the supergroup.

It follows from this that a data receiver can detect any change to any module of a download scenario by just monitoring the transaction_id of the top-level control message of the scenario; i.e., when any module of the download scenario changes, the version subfield of the transaction_id of the top-level control message changes. It works like this: Whenever a module changes in any way, the version number of the module changes. This causes a change in the DII message referencing the module, so the version subfield of the transaction_id of that DII message changes as well. In a two-layer download scenario, this causes a change in the DSI message for the scenario, so the version subfield of its transaction_id changes.

6.1.3 Asynchronous Data Modules in the Data Download Protocol

The non-flow control scenario embodies the unidirectional, one time transmission of a bounded data image to a data receiver. The data carousel cyclically repeats the contents of the carousel, one or more times. If a data 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. Because

the non-flow controlled scenario is a sub-set of the data carousel, for the remainder of this section references to the data carousel imply a reference to the non-flow controlled delivery case as well.

Figure 6.2 illustrates a data carousel implementation of the Download protocol.

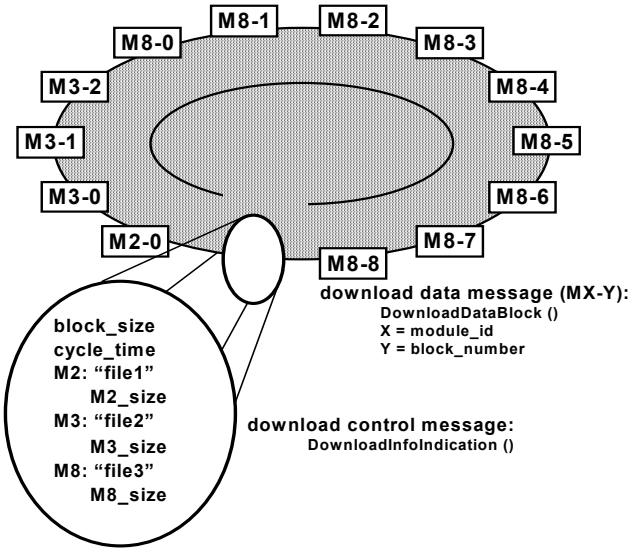


Figure 6.2 Cyclic transmission of information in a data carousel.

Within a data carousel, the data is structured into modules, depicted (for example) in Figure 6.1 as M2, M3 and M8. This abstraction could represent the contents of a number of files, say "file1", "file2" and "file3" as in this example, or the module could be a complex data object. In both cases, the definition of module is outside the scope of ATSC Data Broadcast Standard [1] and of this document. Each module is divided to form the payload of one or more download data messages each defined using the DSM-CC DownloadDataBlock (DDB) 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 the DSM-CC DownloadInfoIndication (DII) and DownloadServerInitiate (DSI) syntax. A third control message, DownloadCancel (DC), can be used to signal the end of a download.

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

The descriptions of the modules in the download protocol are provided by DownloadInfoIndication messages. All the modules comprising a download scenario must be announced in DownloadInfoIndication messages. An additional organizational layer is enabled through the use of DownloadServerInitiate messages that provides a grouping capability for modules.

6.1.4 Asynchronous Data Streaming in the Data Download Protocol

Asynchronous data streaming services are carried via the DSM-CC Download protocol and are identified by the module size being unspecified (zero value). The moduleId, moduleSize, moduleVersion and moduleInfoLength fields associated with a data module conveying asynchronous streaming data must appear in one of the DownloadInfoIndication messages and the value of the moduleSize field must be set to 0. An example of the use of the asynchronous data streaming might be the transmission of a stock ticker service or weather information.

6.1.5 Non-Streaming Synchronized Data Modules (Synchronized Data Download Protocol)

The non-flow controlled scenario of the data download protocol may also be used to convey non-streaming, synchronized data modules. This mode is supported by adding time information in the DSM-CC adaptation header in the DownloadDataBlock message as specified in [1]. The resulting protocol is called the Synchronized Download protocol. Time information in the DSM-CC adaptation header is in the form of a Presentation Time Stamp (PTS) as defined in [3]. It is recommended that the section_syntax_indicator bit be set to 0 and the checksum field be set to 0x0000 to indicate that the checksum has not been calculated. The purpose of this recommendation is to facilitate insertion of time-stamped DSM-CC sections into the ATSC multiplex (or modification of an existing time-stamp in a synchronized module) by not requiring the multiplexer to (re)calculate either a checksum or CRC32.

The payload_unit_start_indicator field in any MPEG-2 Transport Stream packet conveying the beginning of a DSM-CC section containing non-streaming, synchronized data is set to 1. The value of the pointer_field must always be set to 0. This condition indicates that the DSM-CC section starts immediately after the pointer_field. The purpose of this constraint is to fix the position of the PTS in the Transport Stream packets allowing the remultiplexing equipment to modify the time stamp more easily. In addition, it is recommended that the transport packet adaptation field not be used, in order to further ensure that the position of the PTS be located at a fixed offset from the start of the packet. This recommendation results in a value for the adaptation_field_control of 0x01, signifying "No adaptation_field, payload only". Furthermore in this case, an MPEG-2 Transport Stream packet must not include the beginning of more than one DSM-CC section. The transmission of all sections belonging to a single synchronized data module must be finished before the transmission of the next synchronized data module is allowed to start.

6.1.6 Data Download Protocol Hierarchy

A download scenario will be encapsulated using the DSM-CC User-To-Network Download protocol as represented in Figure 6.3.

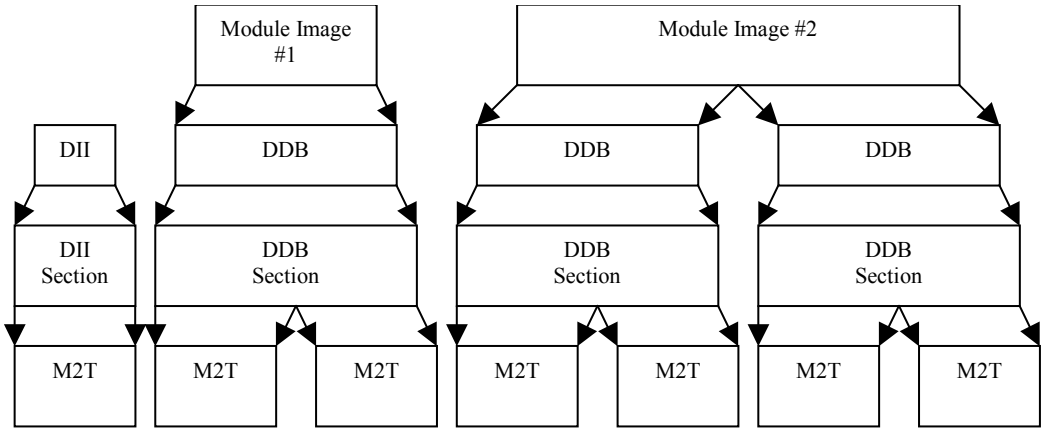


Figure 6.3 Data download protocol module fragmentation.

The DownloadInfoIndication (DII) message(s) publishes all the modules belonging to a Group. The DII is encapsulated into one DSMCC_section and then fragmented into one or more MPEG-2 Transport Stream packets (M2T). In the case where the data module listing information is too large to fit within the payload of a single DSMCC_section, then the two-layer download scenario mechanism must be used. (A two-layer download scenario incorporates a DownloadServerInitiate (DSI) message describing a superGroup, where each of the Groups in a superGroup is described by its own DII message.)

The encapsulated images are first segmented into one or more DownloadDataBlocks (DDBs). In turn, the DDBs are encapsulated into DSMCC_sections and then fragmented into one or more MPEG-2 Transport Stream packets identified by a common elementary_PID value.

All data download protocol messages, DSMCC_sections, and MPEG-2 Transport Stream packets of the same download scenario will be placed in the same Transport Stream Packet Identifier (PID) except for the PSI/SDF/PSIP information as required by the ATSC Data Broadcast Standards [1] and [6].

6.1.7 Data Download Protocol in an MPEG-2 Transport Stream Packet

Each MPEG-2 Transport Stream packet will contain a portion of the data download protocol encapsulation. The MPEG-2 Transport Stream packet header for those MPEG-2 Transport Stream packets containing the start of a DSMCC_section is defined as follows:

Table 6.2 MPEG-2 Transport Stream Packet Header for DSM-CC Section

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	
transport_error_indicator	1		Set in the transmission system to indicate an error in the current Transport Stream packet.
payload_unit_start_indicator (PUSI)	1		Set to 1b[1] in the first MPEG-2 Transport Stream packet and 0b in all remaining MPEG-2 Transport Stream packets that comprise the encapsulated section. In the case of back-to-back sections, this bit will also be set when the new section starts in the MPEG-2 payload.
transport_priority	1	0b	
PID	13		
transport_scrambling_control	2		00b = Not scrambled. See MPEG-2 Systems [2] for additional choices.
adaptation_field_control	2		01 = No adaptation field, payload only. See MPEG-2 Systems [2] for additional choices.
continuity_counter	4	0x0 to 0xF	Each successive packet on a given PID will have the continuity counter incremented over the value carried in the previous packet.
pointer_field	8	0 to 182	Indicates the location of the section start in bytes following this field. This field is only included in the first MPEG-2 Transport Stream packet of the section encapsulation. This field is present when the payload_unit_start_indicator bit is set and the value is the number of bytes until the new section starts in the MPEG payload. When the packet contains the start of a synchronized DDB (Synchronized Download protocol), the pointer field must be set to zero and the payload must start immediately afterwards.
payload	N * 8		The DSMCC_section will be contained here. The length (N) can be up to 183 bytes in a packet when the PUSI bit is set, and up to 184 bytes otherwise. If the final bytes of section are less than the length of the payload area of the MPEG-2 Transport Stream packet, then either a new section may be started assuming the correct pointer field, etc., or the TS packet may be padded out with 0xFF. The Synchronized Download protocol requires that all payloads start immediately following the pointer_field in the MPEG-2 Transport Stream packet. Thus, all payloads must be padded out for this protocol.

6.1.8 DSMCC_section

Each download scenario message (either the DSI, the DII, the DC or the DDB) described below is encapsulated in a DMSCC_section. The DSMCC_section syntax is modeled after the MPEG-2 Transport Stream private_section syntax. The DSMCC_section fields that are included in each message are described below providing a linear look at the entire DSMCC_section that will be constructed.

1 Constructs of the form NMb signify bit fields.

6.1.9 DownloadServerInitiate (DSI)

The DownloadServerInitiate message is used for the two-layer download scenario and it describes each of the Groups in the next layer.

Table 6.3 DSI Message

Field Name	No of Bits	Field Value	Notes
DSMCC_section() {			
table_id	8	0x3B	0x3B = DSM-CC section with DownloadServerInitiate messages.
section_syntax_indicator	1		0b indicates that the checksum/CRC32 field contains a checksum. 1b indicates that the checksum/CRC32 field contains a CRC32.
private_indicator	1		This field is set to the complement of the section_syntax_indicator flag.
reserved	2	11b	
dsmcc_section_length	12		The number of bytes from table_id_extension through the last byte of the CRC32/Checksum field. The maximum value is 4093.
table_id_extension	16		This field is set to the 2 least significant bytes of the transactionId in the dsmccMessageHeader.
reserved	2	11b	
version_number	5		This field is set to 00000b.
current_next_indicator	1	1b	This field is set to 1b, indicating that the information in the DSI is always current.
section_number	8	0x00	This field is set to 0x00.
last_section_number	8	0x00	This field is set to 0x00.
downloadServerInitiate() {			
dsmccMessageHeader() {			
protocolDiscriminator	8	0x11	DSMCC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1006	DownloadServerInitiate message.
transactionId() {			This 32-bit field is split into the sub-fields as indicated below. (See Section 6.1.2 for additional information.)
originator subfield	2	10b	'10' indicating that the server sets this field.
version subfield	14		The value in this field is incremented/changed each time any message in the download scenario is updated.
identification subfield	15	0x0000	'000000000000000' (all zeros). The DSI is a top-level control message.

updated subfield	1		This field is toggled each time the DSI message is updated.
}			End of sub-fields in transactionId.
reserved	8	0xFF	
adaptationLength	8	0x00	This field is set to 0x00, indicating that no adaptation header is present.
messageLength	16		The length of message following this field (from serverID field up to, but not including the checksum/CRC32 field).
dsmccAdaptatinHeader() {			Not used if the adaptation length field is zero.
}			
}			End dsmccMessageHeader().
serverId	20B		This field is set to 20 bytes, each containing the value 0xFF
compatibility_descriptor() {			The compatibilityDescriptorLength field is set to 0x0000, indicating no compatibility_descriptor.
compatibilityDescriptorLength	16	0x0000	Specifies the total length of the descriptors that follow, not including the compatibilityDescriptorLength itself.
}			
privateDataLength	16		Length in bytes of the following GroupInfoIndication structure
groupInfoIndication(){			
numberOfGroups	16		Number of groups in the loop to follow.
for (i=0; i<numberOfGroups; i++) {			
groupId	32		This field is equal to the transactionID of the DII message that describes the group.
groupSize	32		This field will indicate the cumulative size in bytes of all the modules in the group. The size is the sum of all of the moduleSize values in the DII describing this group. This field is set to a value of 0x00000000 when unbounded modules are present in the group.
groupCompatibility{}			The groupCompatibility structure is equal to the compatibility_descriptor structure of DSM-CC [4].
groupInfoLength	16		Length in bytes of descriptor loop that follows.
for (j=0; j<groupInfoLength; j++) {			
groupInfoByte	8		These fields will convey a list of descriptors that each define one or more attributes. The descriptors included in the loop will describe the characteristics of the Group.
}			

}			
groupsInfoPrivateDataLength	16		Length in bytes of the following groupsInfoPrivateData field.
for (j=0; j<groupsInfoPrivateDataLength; j++) {			
groupsInfoPrivateDataByte	8		The contents of this field are user defined.
}			
}			End groupInfoIndication
}			End downloadServerInitiate().
checksum/CRC_32	32		See the CRC32 / Checksum Calculation (6.1.16) below for further information.
}			End DSMCC_section.

6.1.10 DownloadInfoIndication (DII)

There will be at least one DownloadInfoIndication (DII) message. In the case of a two-layer download scenario, the DII describes the modules that are part of the Group. For a one-layer download scenario, the DII must describe all the modules in the download scenario. The full DSMCC_section DII is constructed as follows:

Table 6.4 DII Message

Field Name	No. of Bits	Field Value	Notes
DSMCC_section() {			
table_id	8	0x3B	0x3B = DSM-CC section with DownloadInfoIndication messages.
section_syntax_indicator	1		0b indicates that the checksum/CRC32 field contains a checksum. 1b indicates that the checksum/CRC32 field contains a CRC32.
private_indicator	1		This field is set to the complement of the section_syntax_indicator flag.
reserved	2	11b	
dsmcc_section_length	12		The number of bytes from table_id_extension through the last byte of the checksum/CRC32 field. The maximum value is 4093.
table_id_extension	16		This field is set to the 2 least significant bytes of the transactionId in the dsmccMessageHeader.
reserved	2	11b	
version_number	5		This field is set to 00000b.
current_next_indicator	1	1b	This field is set to 1b, indicating that the information in the DII is always current.
section_number	8	0x00	This field is set to 0x00.
last_section_number	8	0x00	This field is set to 0x00.
downloadInfoIndication() {			
dsmccMessageHeader() {			

protocolDiscrimiator	8	0x11	DSMCC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1002	DII.
transactionId(){			This 32-bit field is split up into a number of sub-fields as indicated below. (See Section 6.1.2 for additional information.)
originator subfield	2	10b	'10' indicating that the server sets this field.
version subfield	14		The value in this field is incremented/changed each time the DII message is updated[2].
identification subfield	15		For a one-layer download scenario, '000000000000000' (all zeros). For a two-layer download scenario, this sub-field is used to identify the DII message (binding to the groupId in the DSI).
updated subfield	1		This field is toggled each time the DII message is updated.
}			End of sub-fields in transactionId
reserved	8	0xFF	
adaptationLength	8	0x00	This field is set to 0x00 to indicate that no adaptation header is present.
messageLength	16		Length of message immediately following this field (including the dsmccAdaptationHeader as specified by the adaptationLength) up to, but not including the checksum/CRC32 field.
dsmccAdaptionHeader() {			Not used if the adaptation length field is zero.
}			
}			
}			End dsmccMessageHeader().
downloadId	32		In the case of a data carousel of the data download protocol, this field is set to be equal to the value of carousel_id if it is specified in the download_descriptor in the associated Data Service Table (DST). The downloadId field is used to associate the download data messages and the download control messages of a single instance of a download scenario.
blockSize	16		Length of data in every block of the DDB except the last. This value is typically 4066 bytes for non-synchronized downloads (stream type 0x0B). (Range 1-4066.). For synchronized downloads (stream type 0x14), the typical value is 4058B.
windowSize	8	0x00	Not used in broadcast.

2 Note: A change in a module in the download scenario will necessitate a change in the version sub-field, since the moduleVersion field in the moduleInfoBytes loop must be updated.

ackPeriod	8	0x00	Not used in broadcast.
tcDownloadWindow	32	0x00000000	Not used in broadcast.
tcDownloadScenario	32		Timeout period in microseconds for entire download. (Optional, use 0x00 if not used).
compatibility_descriptor() {			The descriptorLength field is set to 0x0000. Thus, the compatibility_descriptor structure will always be 2 bytes long.
compatibilityDescriptorLength	16	0x0000	Specifies the total length of the descriptors that follow including the descriptorCount but not including the compatibilityDescriptorLength itself.
}			
numberOfModules	16		Number of modules in the loop to follow.
for (i=0; i<numberOfModules; i++) {			
moduleId	16	0x00 to 0xFFEF	Unique identifier for the module. The values 0xFFF0 to 0xFFFF are reserved by DAVIC.
moduleSize	32		Size of the module in bytes when bounded. Set to 0 when the module size is unspecified. The size is interpreted to be the sum of the payload of each DownloadDataBlock, not counting header syntax or checksum/CRC32.
moduleVersion	8		Version number of the module.
moduleInfoLength	8	0 to 255	Length in Bytes of the loop to follow
for (j=0; j<moduleInfoLength; j++) {			
descriptor()			For moduleID values from 0x0000-0xFFEF, MPEG descriptors such as those listed in Section 6.1.10.1 may be used to provide additional information about a given module.
}			
}			End for of numberOfModules.
privateDataLength	16		Length in Bytes of User Private Data
for (i=0; i<privateDataLength; i++) {			
privateDataByte	8		User private data.
}			
}			End DownloadInfoIndication().
checksum/CRC_32	32		See the CRC32 / Checksum Calculation (6.1.16) below for further information.
}			End DSMCC_section.

6.1.10.1 DII and DSI Descriptors

The following descriptors can be used within the DSI or DII messages (as indicated) to extend the definition of the download scenario.

Table 6.5 II and DSI Descriptors

Descriptor	Tag value	DII - moduleInfo	DSI - groupInfo	Short Description
Reserved	0x00			
module_link_descriptor	0xB4	+		Concatenated data module
CRC32_descriptor	0xB5	+		Cyclic Redundancy Code
group_link_descriptor	0xB8		+	Links DII messages describing a Group

6.1.10.1.1 Module Link Descriptor

The module_link_descriptor contains information regarding those modules that are to be linked together to acquire a complete piece of data from the download scenario. The descriptor also informs the data receiver of the order of the linked modules. The syntax of the module_link_descriptor is shown in Table 6.6.

Table 6.6 Module Link Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
module_link_descriptor () {			
descriptor_tag	8	0xB4	
descriptor_length	8		Number of bytes remaining in the descriptor.
position	8		The position of this module in the chain. The value of 0x00 will indicate the first module of the list. The value of 0x01 indicates an intermediate module in the list and the value of 0x02 indicates the last module of the list.
module_id	16		This is a 16-bit field that identifies the next module in the list. This field is ignored for the last value in the list.
}			

6.1.10.1.2 CRC32 Descriptor

The CRC32_descriptor indicates the calculation of a CRC32 over a complete module. The descriptor can be used as an optimization for providing the CRC32 over the complete module instead of /or in conjunction with the CRC32 associated with each DSMCC_section. Table 6.7 shows the syntax of the CRC32_descriptor.

Table 6.7 CRC32 Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
CRC32_descriptor () {			
descriptor_tag	8	0xB5	
descriptor_length	8		Number of bytes remaining in the descriptor.
CRC_32	32		This is a 32-bit field that contains the CRC calculated over the entire module, which is calculated according to Annex B of the MPEG 2 systems specification [2].
}			

6.1.10.1.3 Group Link Descriptor

The group_link_descriptor contains information regarding which Group descriptions are to be linked in order to describe a single larger Group. This is necessary when the description of modules in a Group exceeds the maximum size of a single DownloadInfoIndication message and has to be spread across a number of such messages. It also informs the decoder on the order of the linked Group descriptions. This descriptor is not strictly necessary as the order of linking is not important. It purely provides a means to identify all the Group descriptions that are linked. The syntax of the group_link_descriptor is shown in Table 6.8.

Table 6.8 Group Link Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
group_link_descriptor () {			
descriptor_tag	8	0xB8	
descriptor_length	8		Number of bytes remaining in the descriptor.
position	8		This 8-bit field identifies the position of this Group description in the chain. The value of 0x00 will indicate the first Group description of the list. The value of 0x01 indicates an intermediate Group and the value of 0x02 indicates the last Group description.
group_Id	32		This 32-bit field identifies the next Group description in the list. This field is ignored for the last value in the list.
}			

6.1.11 DownloadDataBlock (DDB)

The DownloadDataBlock (DDB) is used to encapsulate the data contained in the modules. The full DSMCC_section encapsulated DDB is constructed as follows:

Table 6.9 DDB Message

Field Name	No. of Bits	Field Value	Notes
DSMCC_section() {			
table_id	8	0x3C	0x3C = DSM-CC section with Download Data messages.
section_syntax_indicator	1		0b indicates that the checksum/CRC32 field contains a checksum. 1b indicates that the checksum/CRC32 field contains a CRC32.
private_indicator	1		This field is set to the complement of the section_syntax_indicator flag.
reserved	2	11b	
dsmcc_section_length	12		The number of bytes from the start of table_id_extension through the last byte of the CRC32/Checksum field. The maximum value is 4093 indicating a maximum payload value of 4066 bytes. (Range 1-4066.)
table_id_extension	16		This field will convey a copy of the moduleId value found in the DDB.
reserved	2	11b	
version_number	5		This field will convey a copy of the least significant 5-bits of the moduleVersion value found in the DDB.
current_next_indicator	1	1b	The data is always current.
section_number	8		This field will convey a copy of the least significant 8-bits of the blockNumber value found in the DDB.
last_section_number	8		This field is set to the maximum value that is encoded in the section_number field for the same table_id, table_id_extension and version_number fields.
downloadDataMessage() {			
dsmccDownloadDataHeader() {			
protocolDiscriminator	8	0x11	DSMCC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1003	DDB
downloadId	32		Identifier of the download scenario in progress. In the case of a data carousel scenario of the data download protocol, this field is set to the value of the carousel_id if it is specified in the download_descriptor in the associated Data Service Table.
reserved	8	0xFF	
adaptationLength	8	0x00 or 0x08	Set to 0x08 when a PTS field is present in the adaptation header and set to 0x00 when no PTS field is present. The length must be a multiple of four bytes.

messageLength	16		Length of the message immediately following this field (including the dsmccAdaptationHeader as specified by the adaptationLength) up to, but not including the checksum/CRC32 field.
if (adaptionLength > 0) {			
dsmccAdaptionHeader() {			
adaptationType	8	0x00 or 0x04	If 0x04 then a PTS is present.
if (adaptionType == 0x04) {			
reserved	16	'1…1'b	
'0010'	4		Same as in PES header so PTS information is byte aligned.
PTS[32…30]	3		
marker_bit	1	1b	
PTS[29…15]	15		
marker_bit	1	1b	
PTS[14…0]	15		
marker_bit	1	1b	
}			
}			End dsmccAdaptionHeader().
}			
}			End dsmccDownloadDataHeader().
moduleId	16		The moduleId field identifies to which module this block belongs. The moduleId is an identifier for the module that is described by the moduleSize, moduleVersion, and moduleInfoByte fields. The values 0xFFF0 to 0xFFFF are reserved by DAVIC.
moduleVersion	8		
reserved	8	0xFF	
blockNumber	16		Position of block with the module. The field's numbering starts with zero, and zero corresponds to the first block.
for(i=0;i<N;i++) {			N is the difference between the messageLength minus the adaptationLength minus the 6 bytes of static field from the moduleId through the blockNumber.
blockDataByte	8		Actual data from file/module.
}			
}			End downloadDataMessage().
checksum/CRC_32	32		See the CRC32 / Checksum Calculation (6.1.16) below for further information.
}			End DSMCC_section().

When the DSMCC_section of the DDB is carrying non-streaming, synchronized Download protocol data (stream_type 0x14) it is recommended that the section_syntax_indicator bit is set to 0b

and the checksum field be set to 0x00000000. The purpose of this recommendation is to facilitate the insertion of time-stamped DSM-CC sections into the ATSC multiplex by not requiring the multiplexer to calculate either a checksum or CRC32 once the section has been time-stamped.

6.1.12 DownloadCancel (DC)

The DownloadCancel control message provides a means to signal the cancellation of a download scenario that is already in progress. Table 6.10 illustrates the full semantics of the DownloadCancel message.

Table 6.10 DC Message

Field Name	No. of Bits	Field Value	Notes
DSMCC_section() {			
table_id	8	0x3B	0x3B = DSM-CC section with DownloadCancel message.
section_syntax_indicator	1		0b indicates that the checksum/CRC32 field contains a checksum. 1b indicates that the checksum/CRC32 field contains a CRC32.
private_indicator	1		This is a complement of the section_syntax_indicator flag.
reserved	2	11b	
dsmcc_section_length	12		The number of bytes from the start of table_id_extension through the last byte of the CRC32/Checksum field. The maximum value is 4093 indicating a maximum payload value of 4066 bytes. (Range 1-4066.)
table_id_extension	16		This field is set to the 2 least significant bytes of the transactionID in the dsmccMessageHeader.
reserved	2	11b	
version_number	5		This field is set to 00000b.
current_next_indicator	1	1b	Data is always current.
section_number	8	0x00	This field is set to 0x00.
last_section_number	8	0x00	This field is set to 0x00.
downloadCancel() {			
dsmccDownloadDataHeader() {			
protocolDiscriminator	8	0x11	DSMCC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1005	DownloadCancel.
transactionId {			Sub-fields as defined for DII and DSI messages, except that the downloadCancel message is considered a control message, so the identification sub-field must contain all zero's.
originator subfield	2	10b	'10' indicating that the server sets this field.

version subfield	14		The value in this field is incremented/changed each time any message in the download scenario is updated.
identification subfield	15	0x0000	'000000000000000' (all zeros). The DC is a top-level control message.
updated subfield	1		This field is toggled each time the DC message is updated.
}			End transactionId
reserved	8	0xFF	
adaptationLength	8	0x00	Set to 0x00.
messageLength	16		Length of message immediately following this field (including the dsmccAdaptationHeader as specified by the adaptationLength) up to, but not including the checksum/CRC32 field.
}			End dsmccDownloadDataHeader().
downloadId	32		Identifier of the instance of the download scenario in progress. Provides binding to a particular scenario through the transactionId sub-fields.
moduleId	16		The last processed DownloadDataBlock message at the time of the cancel. If no data blocks have been processed, this field is set to zero.
blockNumber	16		The last processed DownloadDataBlock message at the time of the cancel. If no data blocks have been processed, this field is set to zero.
downloadCancelReason	8		This field contains a code that explains the reason for the cancellation (as specified in [4]).
reserved	8	0xFF	
privateDataLength	16		The length in bytes of the following privateDataByte fields.
for(i=0;i<N;i++) {			N is the difference between the messageLength minus the adaptationLength minus the 6 bytes of static field from the moduleId through the blockNumber.
privateDataByte	8		User defined.
}			
}			End downloadCancelMessage().
checksum/CRC_32	32		See the CRC32 / Checksum Calculation (6.1.16) below for further information.
}			End DSMCC_section().

6.1.13 PSIP Announcement of the Data Download Protocol

In PSIP, the Virtual Channel Table defines the list of Virtual Channels applicable for an ATSC Transport multiplex. Each Virtual Channel carries a Service Location Descriptor (SLD) with information connecting the PID and stream_type values. A data receiver may use this information

to quickly determine the Program elements. For announcements, every Virtual Channel uses the source_id as a link to/from the program announcement information. For data broadcasting, the program announcement information comes in either of two tables: the Event Information Table (EIT) or the Data Event Table (DET). A Data Service containing a data download protocol Program element may be announced in either an EIT or a DET. (For further information regarding Data Service announcement specifics, see Section 7 of this document.)

6.1.13.1 Data Service Descriptor in PSIP

For any event containing a data download protocol Program element, there must be a data_service_descriptor present in either the Event Information Table (EIT) or the Data Event Table (DET). (Refer to Section 7.1 for further information on the data_service_descriptor.)

6.1.13.2 PID Count Descriptor

An optional PID_count_descriptor may be included in the DET or EIT of an announced Data Service. (Refer to Section 7.2 in this document for additional information.)

6.1.14 Discovery of a Data Download Protocol Program Element

6.1.14.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT)

PSIP and the MPEG-2 PSI tables (PAT and PMT) are used in data broadcasting for service announcement. The PSIP tables describe in advance the start times and the duration of events so that a data receiver knows when to expect the Data Services to occur. The tables also describe the parental ratings, the Data Service profiles, and similar event features. One particular PSIP table, the Virtual Channel Table (VCT), describes the channel structure in a ATSC Transport multiplex and therefore indicates those channels allocated for Data Services. Both, the VCT and the MPEG-2 PMT indicate how to locate the tables comprising the Service Description Framework (SDF).

The SDF identifies the presence of data being delivered and provides the mechanism for the unambiguous discovery of download data within the current Transport Stream or another Transport Stream using PSIP and the MPEG-2 PSI (PAT and PMT). (Refer to Section 11 of the ATSC Data Broadcast Standard [1], and Section 7 of this document for additional information regarding Data Service announcements. See Section 12 of [1], and Section 8 of this document for additional information regarding the SDF.)

The Service Location Descriptor (SLD) of the VCT contains the stream_type and elementary_PID information for each data download protocol Program element of the Data Service. Additionally, the same information is carried in the PMT of the current Transport Stream. The possible data download protocol stream_type field values associated with the elementary_PID field are listed below:

- For the delivery of bounded asynchronous data modules, the stream_type field of the SLD and the PMT is 0x0B.
- For the delivery of asynchronous data streaming data download protocol, the stream_type field of the SLD and the PMT is 0x0B.
- For the delivery of synchronized, non-streaming non-flow controlled scenario of the data download protocol (i.e., Synchronized Download protocol), the stream_type field of the SLD and the PMT is 0x14.

The data delivery MPEG-2 Transport Packet Identifier (PID) for the data download protocol is specified by the elementary_PID of the SLD and the PMT. The declared stream_type associated with

the elementary_PID must match the required stream_type value for the desired data download protocol type.

6.1.15 Binding of a Data Download Protocol Program Element

The Service Description Framework (SDF) provides the binding mechanism for an ATSC Data Broadcast system. The SDF's Data Service Table (DST) must be sent at least once for any ATSC Data Service. The following information must be included as part of the DST. (See Section 8 for additional information regarding the DST.)

6.1.15.1 Protocol Encapsulation

The protocol_encapsulation field is located in the Data Service Table (DST) and identifies the protocol encapsulation used by the Program element:

- The protocol_encapsulation field of the DST is set to 0x01 for the delivery of bounded asynchronous data modules.
- The protocol_encapsulation field of the DST is set to 0x01 for the asynchronous data streaming data download protocol.
- The protocol_encapsulation field of the DST is set to 0x02 for the synchronized, non-streaming non-flow controlled scenario of the data download protocol (i.e., Synchronized Download protocol).
- The protocol_encapsulation_field of the DST is set to 0x0D for the asynchronous carousel scenario of the data download protocol.

6.1.15.2 Download Descriptor

The download_descriptor should be included as part of the SDF binding when the protocol encapsuation field is 0x01, 0x02 or 0x0D. The download_descriptor is located in the descriptor loop of the Tap structure in the Data Service Table (DST). The download_descriptor is cross-referenced by the data download protocol's DownloadInfoIndication message through the download id field. The download_descriptor provides the identification value for the download scenario along with the carousel period if a data carousel is being employed.

The DST must be delivered on a separate PID from the data being delivered by the data download protocol. The DST delivery PID (i.e., the elementary_PID) is specified in the Service Location Descriptor (SLD) of the Virtual Channel Table (VCT) and in the Program Map Table (PMT) using the stream_type reference of 0x95.

Table 6.11 illustrates the syntax of the download_descriptor.

Table 6.11 Download Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
download_descriptor() {			
descriptor_tag	8	0xA6	
descriptor_length	8		
for(i=0;i<N;i++) {			N is the value of descriptor_length.
download_Id	32		Unique ID for the download scenario.
carousel_period	32		0x00000000 indicates the carousel time period is unspecified. 0x00000001 to 0xFFFFFFFE specify the time period in milliseconds. 0xFFFFFFFF indicates that the carousel period is not applicable. This field is set to 0 for the asynchronous non-flow controlled and synchronized download protocol.
DIImsg_time_out_value	32		0x00000000 specifies that there is no timeout period. Otherwise, the value is in milliseconds.
}			
}			

The download_descriptor's carousel_period field is set to 0 to indicate the synchronized, non-flow controlled scenario of the data download protocol (i.e., Synchronized Download protocol).

6.1.16 DSMCC_section CRC32 / Checksum Calculation

The checksum/CRC32 field in the DSMCC sections encapsulating the DSI, DII and DC messages provide a means for error detection. This field may carry a CRC32, checksum or no protection at all. The section_syntax_indicator field is used to differentiate between a CRC32 (field value of 1b) or a checksum (field value of 0b). Furthermore, as mentioned below, if a checksum is signalled, but the checksum field contains a value of 0, then this further signals that no error detection has been provided. The private_indicator must carry the complement of the section_syntax_indicator field (i.e., 0b for CRC32, 1b for checksum).

6.1.16.1 CRC32 Calculation

The CRC32 is calculated over the entire DSMCC_section (from the table_id field to the end of the checksum/CRC32 field) according to Annex B of the MPEG 2 systems specification [2].

6.1.16.2 Checksum Encoding and Decoding

A 32-bit checksum is calculated over the entire DSMCC_section (from the table_id field to the end of the checksum/CRC_32 field). The checksum is calculated by treating the DSMCC_section as a sequence of 32-bit integers and performing one's complement addition over all the integers, most significant byte first, then taking the one's complement of the result. For the purpose of computing the checksum, the value of the checksum field is considered to be 0. If the message length is not a multiple of four bytes, the message is considered padded with zeroed bytes immediately before the first byte of the checksum to bring it up to a multiple of four bytes, for

the purpose of checksum calculation only (these padding bytes are removed after the calculation). If the computed result is 0, then the result is set to 0xFFFFFFFF (the alternative value for a one's complement representation of 0). In cases where a checksum is not desired, the value of the checksum field is set to 0 to indicate the checksum has not been calculated. This feature is useful for networks where error detection is provided at a protocol layer lower than the MPEG-2 Transport Stream. The checksum field is present when the section_syntax_indicator is set to 0b.

The checksum is checked on a received section by doing exactly the same calculation, but using the actual transmitted value of the checksum field in the calculation, instead of using the value 0 for that field. If the result comes out 0x00000000 or 0xFFFFFFFF (alternative representations of 0 in one's complement arithmetic), then no errors exist in the message.

The DSM-CC standard has a slight error in its description of this checksum calculation. It indicates in a parenthetical note that one's complement addition is equivalent to an "exclusive or" or XOR operation. This is not in fact correct. True one's complement addition must be used. (One's complement addition works just like ordinary two's complement addition, except that if an overflow occurs, the result is incremented by one. One can think of this as shifting the overflow bit around and adding it in as a lowest order bit, instead of just throwing it away.)

Also, the description of the checksum calculation given here is slightly different from that in the DSM-CC standard, with regard to the handling of the padding bytes. The description given here is entirely equivalent to that in the DSM-CC standard in terms of the result obtained for the checksum, and it has the advantage that the descriptions of encoding and decoding are entirely symmetrical.

6.2 The ATSC DSM-CC Addressable Section

The ATSC DSM-CC addressable section encapsulation protocol is specified in Section 8 of the ATSC Data Broadcast Standard [1].

6.2.1 Overview

The DSM-CC addressable section encapsulation provides a mechanism for transporting any network protocol using one or more MPEG-2 Transport Stream packets in an ATSC network. A DSM-CC addressable section can carry any network protocol by using the Logical Link Control (LLC) protocol and additionally, if required by the LLC encapsulation, the Sub Network Access Protocol (SNAP) to indicate the network protocol being transported. The ATSC DSM-CC addressable section encapsulation allows for the use of an optional IP escape sequence, allowing IP network traffic to dispense with the LLC encapsulation. The LLCSNAP_flag being set to zero in the DSM-CC addressable section header signals the IP escape sequence. Use of the IP escape sequence is the ATSC preferred method for transporting IP datagrams.

The DSM-CC addressable section encapsulation covers unicast (datagrams targeted to a single receiver), multicast (datagrams targeted to a group of receivers) and broadcast (datagrams targeted to all receivers) datagrams. The DSM-CC addressable section contains in its header a 48-bit device identification address, deviceId, (a.k.a., MAC address) which can be used for addressing individual receivers or groups of receivers. The ATSC does not specify how the device identification addresses are allocated to the receivers.

Due to the broadcast nature of ATSC networks, security of the data is important. The DSM-CC addressable section encapsulation format allows for secure transmission of data by supporting encryption of the datagrams and the dynamic changing of the device identification

address. However, datagram security and encryption are outside the scope of both [1] and this document.

6.2.2 LLC Encapsulation

The ISO/ANSI/IEEE standard on Logical Link Control (LLC) in local and metropolitan area networks [8] describes the LLC encapsulation including the LLC header along with a Sub Network Access Protocol (SNAP) header as a method to identify and transport network protocol data units (PDU). IETF RFC1042 [9] specifies how the header fields are applied to an IP datagram.

In LLC Type 1 operation, unacknowledged connectionless mode, the LLC header is three bytes long and consists of a one byte Destination Service Access Point (DSAP) field, a one byte Source Service Access Point (SSAP) field, a one byte Control field followed by the datagram (information bytes) as shown in Figure 6.4.

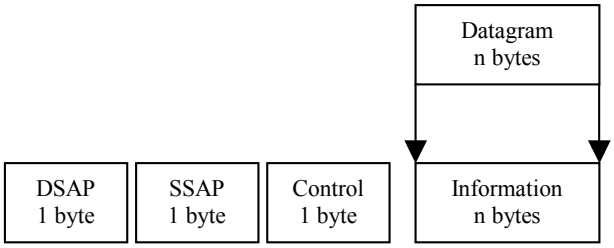


Figure 6.4 LLC encapsulation.

The values 0xAA in the LLC header's DSAP and SSAP fields indicate that an IEEE 802.2 SNAP header follows. The Control value of 0x03 specifies an Unnumbered Information Command PDU. For IP datagrams, the LLC Header value is set to 0xAA-AA-03. Other values of DSAP and SSAP fields indicate support for other network protocols.

The SNAP header is five bytes long and consists of a three byte Organizationally Unique Identifier (OUI) field and a two byte Protocol Identifier (PID – not the MPEG-2 PID) as shown in Figure 6.5.

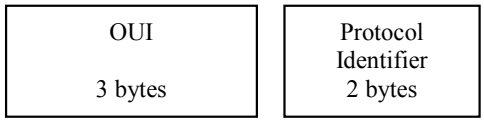


Figure 6.5 SNAP header.

The SNAP OUI value 0x00-00-00 specifies the Protocol Identifier as an EtherType or routed non-OSI protocol. The SNAP OUI of 0x00-80-C2 indicates a Bridged Protocol. When the OUI is set to 0x00-00-00 then the SNAP Protocol Identifier for IP is 0x08-00

Together, the two headers form the LLC/SNAP encapsulation illustrated in Figure 6.6.

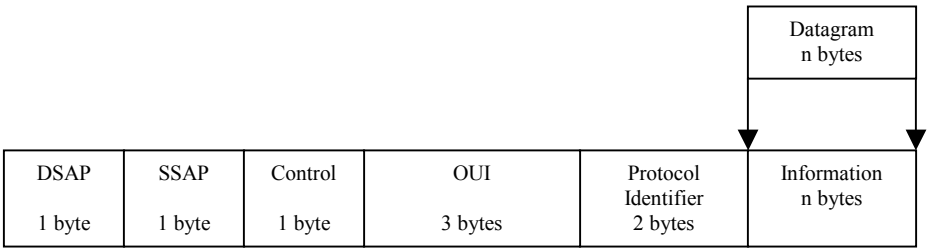


Figure 6.6 LLC/SNAP encapsulation.

For IP datagrams, the complete LLC/SNAP header is 0xAA-AA-03-00-00-00-08-00.

6.2.3 Datagram Transport

Datagrams are transported in a DSMCC_addressable_sections that are compliant with the MPEG-2 private_section syntax format for private data. The section format provides an efficient format for mapping the datagrams to the MPEG-2 Transport Stream packets. The section format supports filtering of datagrams based on the device identification address, deviceId, using existing hardware or software demultiplexers.

Each datagram must fit into a single section[3]. Taking into account the maximum length of a section, this means that the maximum size for an IP datagram is 4072 bytes if LLC encapsulated, and 4080 bytes if the 8-byte LLCSNAP header (part of the LLC encapsulation) is not present (and the recommended method by the ATSC). The maximum size for a datagram of any other protocol is always 4077 bytes or less, since the LLC encapsulation is always required for other network protocols. In other words, the maximum transmission unit (MTU) is 4080 for IP when no LLC encapsulation is used and 4072 when the optional LLCSNAP header (LLC encapsulation) is included for IP, and the MTU is always 4077 bytes or less for any other protocols (which all require the LLC encapsulation). Any protocol datagram larger than the MTU must be fragmented for transmission via DSM-CC addressable sections, using whatever fragmentation mechanisms are available in the network protocol itself. Note: A DSM-CC addressable section may contain only one datagram.

The device identification 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 MPEG-2 private_section while bytes 1, 2, 3 and 4 are in the payload area of the private_section.

3 Some of the language in the ATSC Data Broadcast Standard could be interpreted as implying that datagrams may extend across multiple sections, but Section 8.1 of the standard states flatly that datagrams shall never be split across sections.

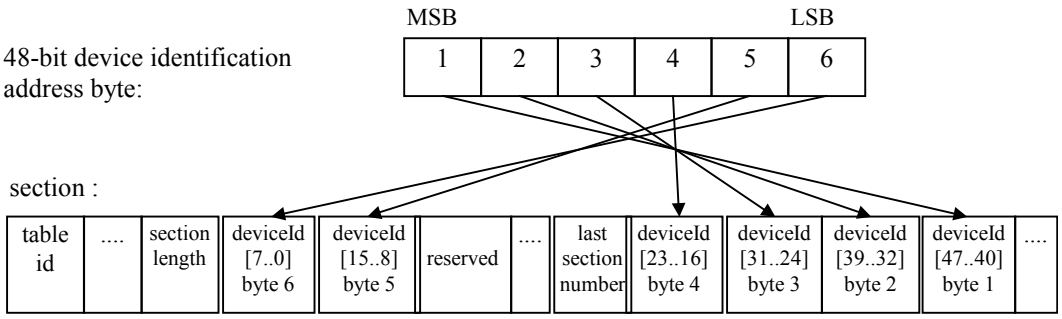


Figure 6.7 Device ID mapping.

Some demultiplexers are able to filter bytes 5 and 6 with hardware. The filtering of bytes 1,2,3 and 4 may be accomplished in either hardware or software. It is recommended that the two bytes of the device identification address that most probably will differentiate the receivers should be placed into deviceId bytes 5 and 6 as shown in Figure 6.7. (This mechanism is normally the case with IEEE MAC addresses, and it is recommended that all MAC addresses be constructed in this way.)

Section scrambling is controlled by the 2-bit field payload_scrambling_control. If the value of these bits is 00b, the payload is not scrambled. If the payload is scrambled, the scrambling algorithm is private to the Data Service. The mechanism describing how the scrambling method is signaled to the receiver is not defined by the ATSC, the ATSC Data Broadcast Standard [1] or this document.

Device identification address scrambling provides further security by dynamically changing the deviceId addresses. By periodically changing the control word that is used for scrambling the deviceId addresses, the monitoring of the stream can be prevented, as the destination of a particular datagram cannot be determined by observing the device identification 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 deviceId addresses is not defined by the ATSC, the ATSC Data Broadcast Standard [1], or this document.

Address scrambling is controlled in the section header by a 2-bit field, the address_scrambling_control. If the value of these bits is 00b, the MAC address is not scrambled. (It should be noted that using device identification 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 DSM-CC addressable 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 break the encryption.

The DSMCC_addressable_section has either a valid CRC_32 or a 32-bit checksum at the end of the section. The checksum field may be set to zero to indicate that no checksum was calculated.

6.2.4 DSM-CC Addressable Section Stream Syntax

Figure 6.8 illustrates the DSMCC_addressable_section encapsulation.

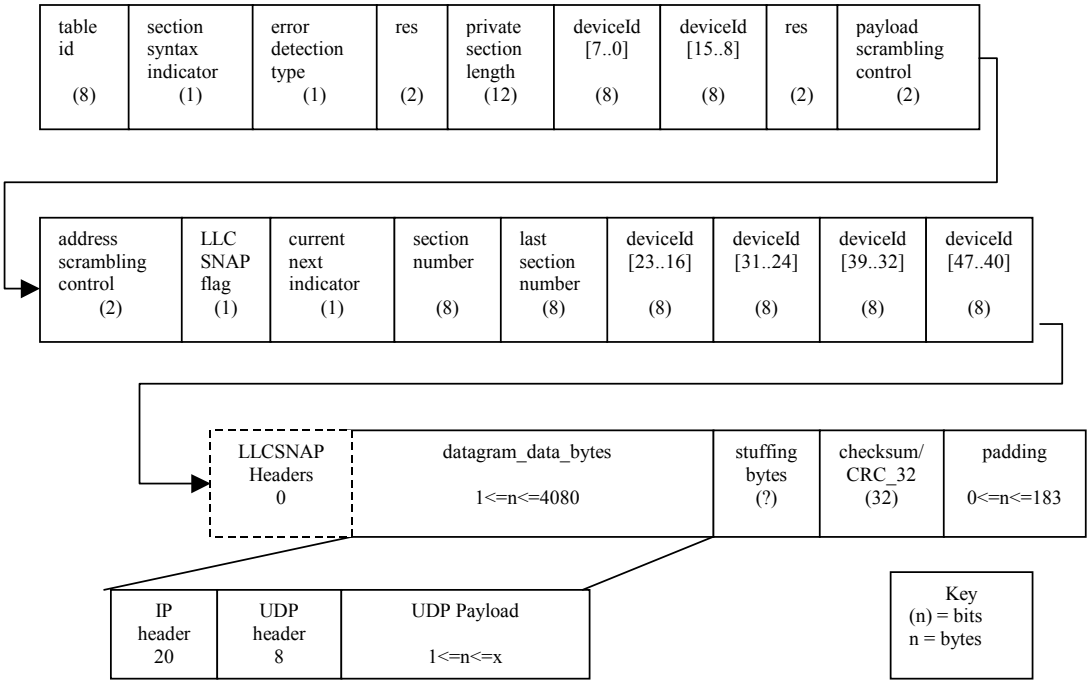


Figure 6.8 DSM-CC addressable section syntax.

6.2.5 Data Transport Specification Usage

The following data structure describes the DSMCC_addressable_section of the ATSC DSM-CC addressable section encapsulation specification used to carry a network datagram. Included in Table 6.12 are the field names, the number of bits the field will consume, the value that is placed in the field if static, and any notes regarding the usage of the individual field.

Table 6.12 DSMCC_addressable_section Syntax

Field Name	No. of Bits	Field Value	Notes
DSMCC_addressable_section() {			
table_id	8	0x3F	
'0'	1	0b	In the ATSC Data Broadcast Standard [1], this field is statically set to 0b to comply with the MPEG-2 System [2] Private Section syntax specification. (Note: In DVB, this field functioned as the section protection control field.)
error_detection_type	1		1b indicates that the checksum/CRC32 field contains a checksum. 0b indicates that the checksum/CRC32 field contains a CRC32. (Note: In DVB, this field was the compliment of the section_syntax_indicator.)
reserved	2	11b	

addressable_section_length	12		The number of bytes from deviceId[7..0] to end of the DSMCC_addressable_section.
deviceId[7..0]	8		Least significant byte of the deviceId (i.e., byte 6). See discussion in Section 6.2.3
deviceId[15..8]	8		Byte 5 of the deviceId.
reserved	2	11b	
payload_scrambling_control	2		00b = Unscrambled. (See the ATSC Data Broadcast Standard [1] for additional values.)
address_scrambling_control	2		00b = Unscrambled. (See the ATSC Data Broadcast Standard [1] for additional values.)
LLCSNAP_flag	1		1b indicates that the datagram is LLC encapsulated. 0b indicates that the datagram is IP and not LLC encapsulated. 0b is used only for IP datagrams, and this value is the ATSC recommended method for carrying IP datagrams. The protocol_encapsulation field of the Data Services Table should be consistent with the setting of this field. The value 0x03 is for LLCSNAP encapsulation and the value 0x04 is for IP without the LLCSNAP encapsulation.
'1'	1	1b	Must be set to 1b.
section_number	8	0x00	Only 1 section allowed for this encapsulation.
last_section_number	8	0x00	Only 1 section allowed for this encapsulation.
deviceId[23..16]	8		Byte 4 of the deviceId.
deviceId[31..24]	8		Byte 3 of the deviceId.
deviceId[39..32]	8		Byte 2 of the deviceId.
deviceId[47..40]	8		Most significant byte of the device Id (i.e., byte 1).
if (LLCSNAP_flag == '1') {			
LLCSNAP()	8 * N		Included if the LLCSNAP_flag is set to 1b. The LLCSNAP() structure contains the LLC header, the optional SNAP header, and the end network portocol datagram.
}			
else {			
for (i=0; i< N1; i++) {			
datagram_data_byte	8		Only an IP datagram data is carried in this space as this field's inclusion is signaled by the IP escape sequence (LLCSNAP_flag == '0')
}			
}			
if(section_number == last_section_number) {			

for (j=0;j<N2;j++) {			
stuffing_bytes	8		Optional field with the value not specified.
}			
}			
checksum/CRC_32	32		See the checksum calculation, Section 6.1.16, in the data download protocol for additional information.
}			

6.2.6 DSM-CC Addressable Section Mapping into MPEG-2 Transport Stream Packets

DSM-CC addressable sections use the MPEG-2 private_section mapping into an MPEG-2 Transport Stream packets. Figure 6.9 shows how a DSM-CC addressable section is encapsulated and mapped into one or more MPEG-2 Transport Stream packets.

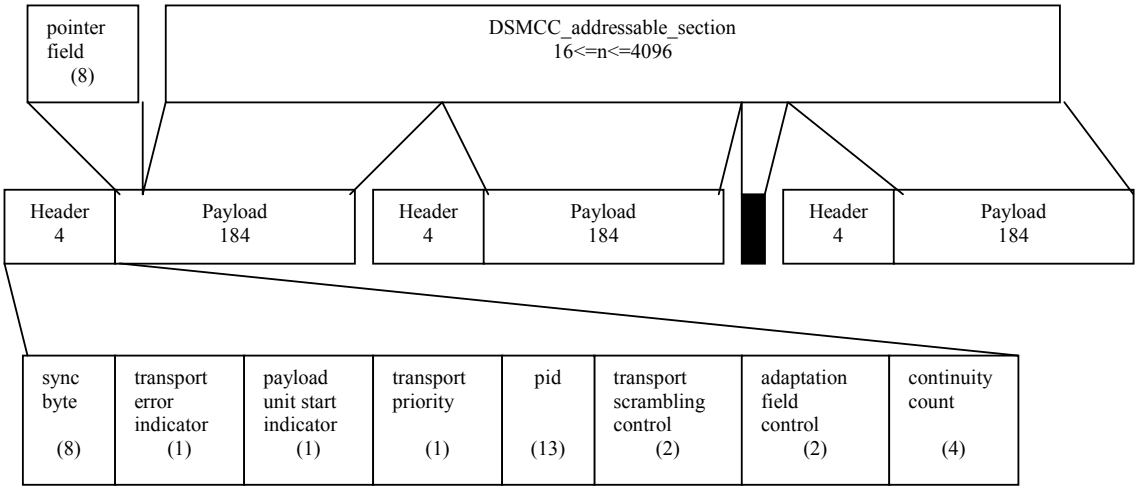


Figure 6.9 A DSMCC_addressable_section encapsulated and mapped into one or more MPEG-2 Transport Stream packets.

A UDP datagram with a payload size of 139 bytes or less that is not LLC encapsulated can be mapped into a single MPEG-2 Transport Stream packet assuming an IP header of 20 bytes and a UDP header of 8 bytes. Any UDP datagram with a payload larger than 139 bytes will require at least two MPEG-2 Transport Stream packets. With LLC encapsulation and with the other previously mentioned assumptions, the maximum payload size of the first MPEG-2 Transport Stream packet is 131 bytes for an IP datagram.

Assuming that a DSM-CC addressable section always begins as the start of an MPEG-2 Transport Stream packet (i.e., not back-to-back sections) and given a 1 byte UDP payload and no LLC encapsulation then the following is the breakdown of the MPEG-2 Transport Stream packet's 184 byte payload: pointer_field (1), DSMCC_addressable_section header (12), UDP/IP header (28), UDP payload (1), checksum/CRC32 (4), and padding bytes (138).

Table 6.13 identifies the fields that comprise an MPEG-2 Transport Stream packet header and MPEG-2 Transport Stream packet payload for a DSM-CC addressable section.

Table 6.13 MPEG-2 Transport Packet Header for DSM-CC addressable section

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	
transport_error_indicator	1		
payload_unit_start_indicator (PUSI)	1		Set to 1b in the first MPEG-2 Transport Stream packet and 0b in all remaining MPEG-2 Transport Stream packets that comprise the encapsulated datagram. In the case of back-to-back sections, this bit will also be set when a new section starts in the MPEG-2 payload.
transport_priority	1		
PID	13		
transport_scrambling_control	2		00b = Not scrambled. (See MPEG-2 Systems [2] for additional values.)
adaptation_field_control	2		01b = No adaptation field, payload only. (See MPEG-2 Systems [2] for additional values.)
continuity_counter	4	0x0 to 0xF	0x0 through 0xF (incremented for each consecutive packet on a given PID).
optional adaptation_field()			Inclusion signaled by the adaptation_field_control.
pointer_field	8		This field is present when the PUSI bit is set and the value is the number of bytes until the new section starts in the MPEG payload. The value 0x00 indicates the section starts immediately following this byte.

6.2.7 PSIP Announcement of the DSM-CC Addressable Section

In PSIP, the Virtual Channel Table defines the list of Virtual Channels applicable for an ATSC Transport multiplex. Each Virtual Channel carries a Service Location Descriptor (SLD) with information connecting the PID and stream_type values. A data receiver may use this information to quickly determine the Program elements. For announcements, every Virtual Channel uses the source_id as a link to the program guide information. For data broadcasting, the program announcement information comes in either of two tables: the Event Information Table (EIT) or the Data Event Table (DET). A Data Service containing a DSM-CC addressable section Program element may be announced in either an EIT or a DET. (For further information regarding Data Service announcement specifics, see Section 7 of this document.)

6.2.7.1 Data Service Descriptor in PSIP

If the ATSC addressable section protocol is an announced event, as per Section 7, then there must be a data_service_descriptor present in either the Event Information Table (EIT) or the Data Event Table (DET). (Refer to Section 7.1 in this document for additional information.)

6.2.7.2 PID Count Descriptor

An optional PID_count_descriptor may be included in the DET or EIT of an announced Data Service. (See Section 7.2 for additional information.)

6.2.8 Discovery of a DSM-CC Addressable Section Program Element

6.2.8.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT)

PSIP and the MPEG-2 PSI tables (PAT and PMT) are used in data broadcasting for service announcement. The PSIP tables describe in advance the start times and the duration of events so that a data receiver knows when to expect the Data Services to occur. The tables also describe the parental ratings, the Data Service profiles, and similar event features. One particular PSIP table, the Virtual Channel Table (VCT), describes the channel structure in a ATSC Transport multiplex and therefore indicates those channels allocated for Data Services. Both, the VCT and the MPEG-2 PMT indicate how to locate the tables comprising the Service Description Framework (SDF).

The SDF identifies the presence of data being delivered and provides the mechanism for the unambiguous discovery of download data within the current Transport Stream or another Transport Stream using the PSIP and the MPEG-2 PSI (PAT and PMT). (Refer to Section 11 of the ATSC Data Broadcast Standard [1], and Section 7 of this document for additional information regarding Data Service announcements. See Section 12 of the ATSC Data Broadcast Standard [1], and Section 8 of this document for additional information regarding the SDF.)

The Service Location Descriptor (SLD) of the VCT contains the stream_type and elementary_PID information for each DSM-CC addressable section Program element of the Data Service. Additionally, the same information is carried in the PMT of the current Transport Stream. The DSM-CC addressable section stream_type field values associated with the elementary_PID field are defined as follows:

> For the delivery of DSM-CC addressable sections, the stream_type field of the SLD and the PMT is 0x0D.

The data delivery MPEG-2 Transport Packet Identifier (PID) for the DSM-CC addressable section protocol is specified by the elementary_PID of the SLD and the PMT. The declared stream_type associated with the elementrary_PID must match the required stream_type value.

6.2.9 Binding of a DSM-CC Addressable Section Program Element

The Service Description Framework (SDF) provides the binding mechanism for an ATSC Data Broadcast system. The SDF's Data Service Table (DST) must be sent at least once for any ATSC Data Service. The following information must be included as part of the DST. (See Section 8 for additional information regarding the DST.)

6.2.9.1 Protocol Encapsulation

The protocol_encapsulation field is located in the Data Service Table (DST) and identifies the protocol encapsulation used by the Program element. The protocol_encapuslation value must match the setting of the LLC_SNAP_flag encapsulation signaling bit located in the DSM-CC addressable section.

- The protocol_encapsulation field of the DST is set to 0x03 for all LLC encapsulated network protocols (i.e. when the LLCSNAP_flag is set to '1').
- The protocol_encapsulation field is set to 0x04 for the non-LLC encapsulated IP protocol (i.e., DSM-CC addressable sections using the IP escape sequence signaled by the LLCSNAP_flag being set to '0').

6.2.9.2 Multiprotocol Encapsulation Descriptor

The multiprotocol_encapsulation_descriptor may optionally be included as part of the SDF's binding information. The multiprotocol_encapsulation_descriptor is located in the descriptor loop of the Tap structure in the DST.

The DST must be delivered on a separate PID from the data being delivered by the DSM-CC addressable section Program element. The DST delivery PID (i.e., the elementary_PID) is specified in the Service Location Descriptor (SLD) of the Virtual Channel Table (VCT) and in the PMT using the stream_type reference of 0x95.

Table 6.14 defines the structure of the multiprotocol_encapuslation_descriptor.

Table 6.14 Multiprotocol Encapsulation Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
multiprotocol_encapuslation_descriptor() {			
descriptor_tag	8	0xA7	
descriptor_length	8	0x02	Description length is 0x02.
deviceId_address_range	3	0x00 to 0x07	0x06 indicates all six bytes are valid. (See the ATSC Data Broadcast Standard [1] for additional values.)
deviceId_IP_mapping_flag	1		1b signals an IP to deviceId (MAC address) mapping per RFC 1112 [7]. 0b signals any device mapping.
alignment_indicator	1	0b	
reserved	3	111b	
max_sections_per_datagram	8	0x01	All datagrams will only use the value 0x01.
}			

The deviceId_address_range field is used for signaling to the data receiver the bytes of the deviceId that are significant for filtering. The significant bytes of the deviceId address begin at the least significant end of the deviceId address.

The deviceId_IP_mapping_flag signals if mapping the multicast IP addresses to deviceId addresses is done according to the RFC 1112 [7]. It should be noted that the ATSC Data Broadcast Standard [1] and this document do not define the deviceId addresses.

The max_sections_per_datagram field defines the maximum number of section that are used for carrying a single datagram (for multiprotocol encapsulation, 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.

6.3 Synchronous and Synchronized Data Streaming

The synchronous and synchronized data streaming encapsulation protocols are specified in Section 9 of the ATSC Data Broadcast Standard [1].

6.3.1 Definitions

A data stream is considered synchronous when a portion of the data packets carries time information that enables the display or the presentation as a strict time-controlled sequence of actions. In addition, a synchronous data stream is composed of 16 bit synchronous data access units (SDAUs) that are leaked to the application in a piecewise continuous rate. In this case, it is said that the clock, carried jointly with the data units, can be generated at the receiver. Synchronous data streams have no timing association with other streams. Furthermore, the data packets of synchronous streams arrive at almost periodic intervals. The word *almost* is used because in practice there is a bound for the minimum and the maximum value that the period may have. An example of a synchronous stream may be a banner that displays the weekend sports results. The banner data will arrive at periodic intervals and it will be displayed also at periodic rate.

A data stream is called synchronized when it has a timing association with a separate component stream (Program element) and consists of Data Access Units that are meant to be "presented" at specific times. An example of a synchronized stream may be an animated GIF image for an electronic coupon. The image may be placed in a corner of the screen at the same time as some specific action happens during a commercial. In this case, the image data is synchronized with the video stream.

Synchronous and synchronized data streams are carried in Packetized Elementary Stream (PES) packets similar to the PES packets that serve to deliver audio and video data for conventional television services.

6.3.2 Encapsulation of PES Packets in Transport Stream Packets

A PES packet is divided into a header and a payload section. The PES-packet header starts with a bit sequence that is normally easy to detect. The PES header sequence is 23 zeros followed by a single one. The next field is the stream_id that is set to 0xBD (private stream 1) for both synchronous and synchronized data streams. The third field in a PES packet header is the packet length. For video streams, the packet length is set to zero to indicate unspecified lengths, however this value (zero) is not allowed for data streams. Consequently, all carried data units (datagrams) have a finite size.

The maximum size of a PES packet is 64 Kbytes (64 Kbytes plus 4 bytes to be more precise) and therefore, it is normally much larger than a Transport Stream packet. The payload of a Transport Stream packet is normally 184 bytes (unless an adaptation field exists), and therefore the PES packet is usually segmented into as many 184-byte units as necessary. A very important requirement of PES encapsulation is that the beginning of a PES packet needs to be always aligned with the beginning of a Transport Stream packet.

In order to align PES packets with Transport Stream packets, it is usually necessary to add enough stuffing bytes to one or more MPEG-2 Transport Stream packets that result from slicing the large PES packet into smaller units. One common stuffing method is to create a dummy adaptation field in the Transport Stream packet header that contains only stuffing bytes. Figure 6.10 illustrates the encapsulation of PES packets in Transport Stream packets.

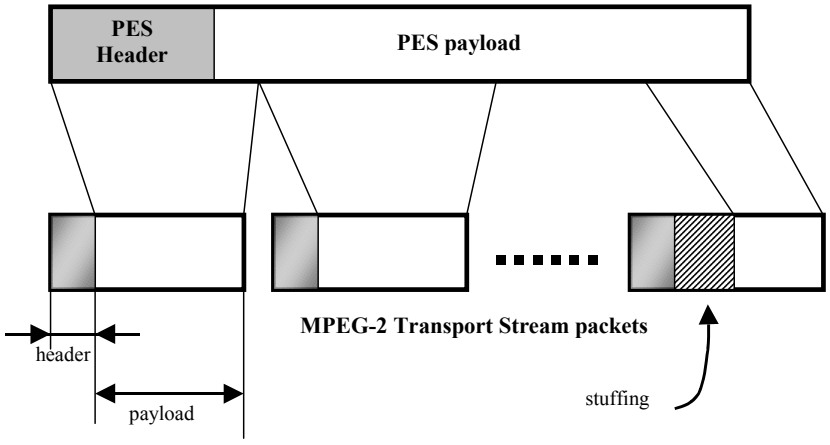


Figure 6.10 Encapsulation of synchronous PES packets in MPEG-2 Transport Stream packets.

6.3.3 Synchronous Data Streaming

A major feature of a synchronous data stream is the timing information carried within the stream itself. A receiver decodes this information and uses it to adjust a local clock that is synchronized to the stream. The PES packet stream_id, set to "private_stream_1" (0xBD), allows for the usage of the PES header fields, especially the Presentation Time Stamp (PTS) fields. For data broadcasting, the accuracy of the PTS needs to be extended beyond the MPEG-2 limits and therefore, an extension to the PTS information, called pts_ext8, is defined and carried in the PES payload. This field extends the resolution of the PTS from 11.1 microseconds to 74 nanoseconds.

The PES packet payload for synchronous data streaming is composed of 16 bit synchronous data access units that are leaked to the application at a piecewise continuous rate.

For a synchronous data stream, the PES payload carries the synchronous_data_header() structure followed by the synchronous_data_sequence( ). The first structure carries two important pieces of information, the PTS extension and possibly a new measure of the stream data rate (using the bit-field called increment). Notice that the stream data rate can be defined in the PES packet header, however the increment field in the synchronous_data_header() structure may override this value.

The synchronous_data_sequence() structure is the collection of bytes that compose a data unit. For synchronous multiprotocol encapsulation, the first 8 bytes of this structure provide the LLC/SNAP field. The method to know if a synchronous data stream carries a LLC/SNAP field is by examining the associated Data Service Table (DST). Entries in the DST indicate the applicable protocol encapsulation via the protocol_encapsulation field. Applicable values for synchronous data streams are listed in Table 6.15.

The following table illustrates synchronous data encapsulated using PES packets.

Table 6.15 Synchronous Data in PES

Field Name	No. of Bits	Field Value	Notes
PES_packet() {			
PES_packet_header() {			Start of PES_packet_header
packet_start_code_prefix	24	0x000001	23 zero bits followed by a one bit. This sequence of zeroes followed by a one was designed to allow a receiver to easily identify the beginning of a PES packet in the bitstream. This assumes of course that this sequence cannot occur at other locations in the bitstream (i.e., start code emulation).
stream_id	8	0xBD	This value indicates: private_stream_1. The PES header bytes of a private_stream_1 stream may include many optional fields including Decoding and Presentation Time Stamps; this is not the case for a private_stream_2-type of stream.
PES_packet_length	16		Length in bytes of remaining PES_packet (from the end of this field to the end of the packet_payload). This value must be non-zero. (The convention for video PES packets of using the value zero to signal an unspecified length is not allowed for synchronous streaming data).
marker_bits	2	10b	Set by MPEG-2 Systems to avoid the possibility of inadvertent start_code emulation.
PES_scrambling_control	2		Indicates the scrambling mode of the PES_packet payload. 00b indicates no scrambling. 01b, 10b and 11b indicate user defined scrambling methods. The PES_packet_header may never be scrambled.
PES_priority	1		Relative priority of PES_packet payload. A ‘1b’ indicates a higher priority for the payload than a 0b. This field is typically set by the multiplexor and should not be changed by the transport mechanism.
data_alignment_indicator	1	0b	0b indicates that the alignment of the Data Access Unit within the PES packet payload is not specified. For ATSC A/90 Data Broadcast Services, this value is typically set to ‘0b’ as alignment rules in MPEG-2 Systems are only defined for video and audio elementary streams.
copyright	1		0b indicates that the content is not protected by copyright. If set to 1b, the copyright method is defined by a copyright descriptor in the PMT of the MPEG-2 Program to which this MPEG-2 PES packet belongs.

original_or_copy	1		1b indicates the content of the PES packet payload is an original. 0b indicates that the content of the PES packet payload is a copy. The purpose of this flag is to enable content rights management and copy protection in the receiver (assuming that these systems have been properly defined and specified).
PTS_DTS_flag	2	10b	10b indicates that the packet header carries a PTS. As discussed below, the DTS does not have any normative meaning for synchronized data and is not commonly used. For synchronous, streaming data, each PES_packet must contain a PTS, so this field will normally carry the value 10b.
ESCR_flag	1		0b indicates that an elementary stream clock reference (ESCR) is not included in the packet header. Elementary Stream clock reference fields are typically used to help re-multiplexing an elementary stream into various MPEG-2 Transport Streams
ES_rate_flag	1		This field indicates the presence of a 22-bit ES_rate field in the PES header. The value of ES_rate specifies the leak rate for the PES packet bytes. The field ES_rate is most commonly used for synchronous data elementary streams. If the data rate is above 9.0 Mbps, then the ES_rate field must be used to indicate the leak rate, thus above 9.0 Mbps, this field must carry the value 1b. If the data rate is below 9.0 Mbps, then the increment field in the synchronous_data_header must be used to indicate the leak rate and any value in the ES_rate field will be ignored.
DSM_trick_mode_flag	1	0b	This field is used to indicate the presence of trick mode control commands in the PES header. This bit is always set to 0b as this functionality is now superseded by the DSM-CC specification.
additional_copy_info_flag	1		This field indicates the presence of additional information regarding copyright of the data in the PES packet (0b indicates that no extra copy information exists in the header). The additional information is represented by the 7-bit additional_copy_info field in the PES packet header.
PES_CRC_flag	1		This field indicates the presence of a CRC16 error detection code applicable to the payload of the previous MPEG-2 PES packet in the same data elementary stream. If set to 1b, the CRC error detection code is carried by the previous_PES_packet_CRC field in the PES packet header applicable to the previous MPEG-2 PES packet.

PES_extension_flag	1	0b	This field indicates whether the PES extension fields are present in the PES packet header or not. The fields in the PES extension are typically relevant to MPEG-2 Program Stream and therefore are not used for ATSC Data Broadcast.
PES_header_data_length	8		This field indicates the length in bytes of the optional fields and stuffing bytes in the PES packet header (all of the fields beginning immediately after this field to the start of the PES_packet_payload). The value of this field is necessary to determine the number of stuffing bytes at the tail of the PES packet header.
If (PTS_DTS_flags == 10b) {			Start of the PTS fields only. The PTS field is a 33-bit field representing a sample of the 90 KHz clock derived from the 27 MHz System Time Clock reconstructed from the PCR fields transmitted in the MPEG-2 Transport Stream packets. The value of the PTS field represents the presentation time of the first data access unit in the PES packet.
'0010'	4	0010b	Required bits when the PES_packet_header carries a PTS, but not a DTS.
PTS [32..30]	3		The 3 most significant bits of the 33-bit Presentation Time Stamp (PTS).
marker_bit	1	1b	Marker bits are always set to 1b.
PTS [29..15]	15		The middle 15-bits of the 33-bit presentation time stamp.
marker_bit	1	1b	Marker bits are always set to 1b.
PTS [14..0]	15		The 15 least significant bits of the 33-bit presentation time stamp.
marker_bit	1	1b	Marker bits are always set to 1b.
}			End of PTS fields only.
if (ES_rate_flag == 1b) {			ES_rate field.
marker_bit	1	1b	Avoids emulation of start bits.
ES_rate	22		Multiplying this 22-bit value by 400 bps, gives the elementary stream rate. Zero is not an allowed value for this field. The ES_rate is used to indicate the leak rate for the synchronous_data_access_units when the data rate is above 9.0 Mbps, otherwise the increment field in the synchronous_data_header is used.
marker_bit	1	1b	Avoids emulation of start bits.
}			End of ES_rate field.
if (PES_CRC_flag == 1b) {			PES_CRC field
previous_PES_packet_CRC	16		CRC16 value of previous PES packet.
}			End of PES_CRC field
for (i=0; i<N2; i++){			stuffing_bytes loop.

stuffing_byte	N2*8	0xFF	The encoder may insert stuffing bytes, which are typically discarded by the receiver. The value of N2 is determined as the difference between the value carried in the PES_header_data_length field and the length of the optional fields up to this point. N2 can't be larger than 32 bytes.
}			End stuffing byte loop.
}			End of PES_packet_header
PES_data_payload() {			Start of PES_data_payload.
synchronous_data_header() {			Start of synchronous_data_header.
pts_ext_8	8		This field contains the 8 most significant bits of the 9 bit Program Clock Reference extension that extends the time resolution of synchronous data PTS from the MPEG-2 standard resolution of 11.1 microseconds (90 kHz) to 74 nanoseconds (13.5 MHz).
data_rate_flag	1		This 1-bit flag indicates the use of the increment field defined below. This field is set to 1b to indicate that an increment field is present in the synchronous data header. This flag must be set to 1b for synchronous data rates between 1.0058 bit/second and 9.0 Mbps. In this case, the ES_rate field, if present in the PES header, will be ignored by the data receiver. This flag must be set to 0b for synchronous data rates greater than 9.0 Mbps. In this case, the data rate shall be specified by the ES_rate field of the PES packet header.
reserved	3	111b	Reserved fields are always filled with 1bs to avoid start_code emulation.
synchronous_data_header_length	4		This 4-bit field indicates the length of the remainder of the synchronous_data_header (from immediately after this field to the start of the synchronous_data_sequence) in units of 16 bit words. The value of this field is "H" in the reserved fields below.
if (data_rate_flag) {			
reserved	4	0xF	Reserved fields are always filled with 1bs to avoid start_code emulation.
increment	28		This field represents the leak rate for the 16 bit synchronous_data_access_units and must be provided if the synchronous data rate is between 1.0058 bits/second and 9.0 Mbps (above 9.0 Mbps, the leak rate is given by the ES rate field in the PES_packet_header. The increment field contains an integer value specifying the ratio of the synchronous data rate to a 27 MHz reference and must be in the range 0x14-0xAAAA6E0.

reserved	16*(H-2)	1…1b	This reserved field is filled in with 1bs to avoid start_code emulation.
else {			
reserved	16*H	1…1b	This reserved field is filled in with 1bs to avoid start_code emulation.
}			
}			End synchronous_data_header.
synchronous_data_sequence() {			Start of synchronous_data_sequence
for (i=0; i<N; i++) {			
synchronous_data_access_unit	16		Synchronous_data_access_units represent the payload of the synchronous, streaming encapsulation and are 16 bit data words that are leaked from the buffer at a piecewise continuous rate specified by either the increment or ES_rate fields.
}			
}			End of synchronous_data_sequence.
}			End of PES_packet_payload.
}			End of PES_packet.

Note: In the formula specified in the ATSC Data Broadcast Standard [1], the constant 27MHz should equal 27,000,000 for calculations.

6.3.4 Synchronous Data Streaming of Network Datagrams

The frame structure defined for synchronous data streams may carry many different data types. One possible use for this structure is the carriage of network datagrams, and more specifically IP datagrams. There are four important facts to remember when encapsulating IP datagrams:

- The use of a LLC/SNAP encapsulation is optional for IP datagrams. However, the ATSC recommends that IP datagrams be carried without using the LLC/SNAP encapsulation. All other network datagrams are required to use the LLC/SNAP encapsulation.
- The Maximum Transfer Unit (MTU) for IP is 4072 bytes when LLC/SNAP is used and 4080 bytes when it is not for all IP datagrams.
- The protocol_encapsulation field attached to the associated tap in the DST should be either 0x08 (LLC/SNAP included) or 0x0A (no LLC/SNAP).
- Only one datagram per PES packet.

Figure 6.11 illustrates the encapsulation of IP datagrams using PES framing for synchronous data streaming.

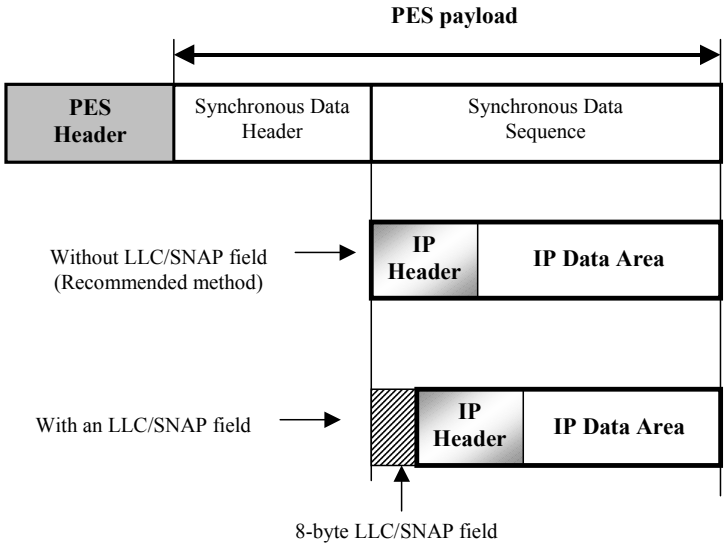


Figure 6.11 Encapsulation of IP packets for synchronous data streaming using PES frames.

6.3.5 Synchronized Data Streaming

Synchronized data streaming is used when a data stream needs time synchronization with one or more separate MPEG-2 PES streams. An example of when this might be applicable is the timed delivery of an IP datagram containing a trigger notification for a separate, pre-delivered graphic to be displayed with associated video and audio PES streams.

The PES packet stream_id field will be set to 0xBD to indicate a type of private_stream_1 allowing for the usage of the PES header fields, especially the Presentation Time Stamp (PTS) fields. The PES_packet_length field must be set to a non-zero value. Usage of the time stamps, i.e. PCR via the adaptation field in an MPEG-2 Transport Stream header, requires the definition of Access Units. In the ATSC Data Broadcast Standard [1], and for PES packetization layer, a Data Access Unit (DAU) is defined as the in-order collection of one or more PES packets starting with the one bearing the PTS and up to (but not including) the next PES packet with a PTS. A PTS value must be present in the first PES packet of every Data Access Unit. The DAU is meant to represent a basic unit of data that is "presented" at the specified PTS.

For a synchronized data stream, the PES payload carries the synchronized_data_packet() structure. This structure starts with a variable-length header followed by the payload bytes. In most cases, the length of this header will be either 3 or 5 bytes. The header bytes may include a 9-bit PTS extension and a privately defined sub-stream identifier. Following the header bytes is the collection of data payload bytes.

Table 6.17 shows PES encapsulation for synchronized data streaming. Notice that the header is similar to the synchronous case and that only the PES payload differs slightly.

Table 6.17 Synchronized Streaming Data in PES

Field Name	No. of Bits	Field Value	Notes
PES_packet() {			
PES_packet_header() {			Start of PES_packet_header
packet_start_code_prefix	24	0x00001	23 zero bits followed by a one bit. This sequence of zeroes followed by a one was designed to allow a receiver to easily identify the beginning of a PES packet in the bitstream. This assumes of course that this sequence cannot occur at other locations in the bitstream (start code emulation).
stream_id	8	0xBD	This value indicates: private_stream_1. The PES header bytes of a private_stream_1 stream may include many optional fields including Decoding and Presentation Time Stamps; this is not the case for a private_stream_2-type of stream.
PES_packet_length	16		Length in bytes of remaining PES_packet (from the end of this field to the end of the packet_payload). This value must be non-zero (the convention for video PES packets of using the value zero to signal an unspecified length is not allowed for synchronized streaming data).
marker_bits	2	10b	Set by 13818-1 to avoid the possibility of inadvertent start_code emulation.
PES_scrambling_control	2		Indicates the scrambling mode of the PES_packet payload. 00b indicates no scrambling. 01b, 10b and 11b indicate user defined scrambling methods. If the payload is scrambled, the PES_packet_header cannot be scrambled.
PES_priority	1		Relative priority of PES_packet payload. A ‘1’ indicates a higher priority for the payload than a 0b. This field is typically set by the multiplexor and should not be changed by the transport mechanism.
data_alignment_indicator	1	0b	0b indicates that the alignment of the Data Access Unit within the PES packet payload is not specified. For ATSC Data Broadcast Services, this value is typically set to ‘0’ as alignment rules in MPEG-2 Systems are only defined for video and audio elementary streams.
copyright	1		0b indicates that the content is not protected by copyright. If set to 1b, the copyright method is defined by a copyright descriptor in the PMT of the MPEG-2 Program to which this PES packet belongs.

original_or_copy	1		1b indicates the content of the PES packet payload is an original. 0b indicates that the content of the PES packet payload is a copy. The purpose of this flag is to enable content rights management and copy protection in the receiver (assuming that these systems have been properly defined and specified).
PTS_DTS_flag	2	10b	10b indicates that the packet header carries a PTS. As discussed below, the DTS does not have any normative meaning for synchronized data and is not commonly used.
ESCR_flag	1		0b indicates that an elementary stream clock reference (ESCR) is not included in the packet header. Elementary Stream clock reference fields are typically used to help re-multiplexing an elementary stream into various Transport Streams
ES_rate_flag	1	0b	This field indicates the presence of a 22-bit ES_rate field in the PES header. The value of ES_rate specifies the leak rate for the PES packet bytes. The field ES_rate is most commonly used for synchronous data elementary streams.
DSM_trick_mode_flag	1	0b	This field is used to indicate the presence of trick mode control commands in the PES header. This bit is always set to 0b as this functionality is now superseded by the DSM-CC Specification.
additional_copy_info_flag	1		This field indicates the presence of additional information regarding copyright of the data in the PES packet (0b indicates that no extra copy information exists in the header). The additional information is represented by the 7-bit field additional_copy_info field in the PES packet header.
PES_CRC_flag	1		This field indicates the presence of a CRC16 error detection code applicable to the payload of the previous PES packet in the same data elementary stream. If set to 1b, the CRC error detection code is carried by the previous_PES_packet_CRC field in the PES packet header.
PES_extension_flag	1	0b	This field indicates whether the PES extension fields are present in the PES packet header or not. The fields in the PES extension are typically relevant to MPEG-2 Program Stream and therefore are not used for ATSC Data Broadcast.

PES_header_data_length	8		This field indicates the length in bytes of the optional fields and stuffing bytes in the PES packet header (all of the fields beginning immediately after this field to the start of the PES_packet_payload). The value of this field is necessary to determine the number of stuffing bytes at the tail of the PES packet header.
if (PTS_DTS_flags == 10b) {			Start of the PTS fields only. The PTS field is a 33-bit field representing a sample of the 90 KHz clock derived from the 27 MHz System Time Clock reconstructed from the PCR fields transmitted in the MPEG-2 Transport Stream packets. The value of the PTS field represents the presentation time of the first data access unit in the PES packet.
'0010'	4	0010b	Required bits when the PES_packet_header carries a PTS, but not a DTS.
PTS [32..30]	3		The 3 most significant bits of the 33-bit Presentation Time Stamp (PTS).
marker_bit	1	1b	Marker bits are always set to 1b.
PTS [29..15]	15		The middle 15-bits of the 33-bit presentation time stamp.
marker_bit	1	1b	Marker bits are always set to 1b.
PTS [14..0]	15		The 15 least significant bits of the 33-bit presentation time stamp.
marker_bit	1	1b	Marker bits are always set to 1b.
}			End of PTS fields only.
if (ESCR_flag == 1b) {			ES_rate field.
marker_bit	1	1b	Avoids emulation of start bits.
ES_rate	22		Multiplying this 22-bit value by 400 bps, gives the elementary stream rate. Zero is not an allowed value for this field.
marker_bit	1	1b	Avoids emulation of start bits.
}			*End of ES_rate field*.
if (PES_CRC_flag == 1b) {			*PES_CRC field*.
previous_PES_packet_CRC	16		CRC16 value of previous PES packet.
}			*End of PES_CRC field.*
for (i=0; i<N2; i++){			*stuffing_bytes loop*.
stuffing_byte	N2*8	0xFF	The encoder may insert stuffing bytes, which are typically discarded by the receiver. The value of N2 is determined as the difference between the value carried in the PES_header_data_length field and the length of the optional fields up to this point. N2 can't be larger than 32 Bytes.
}			End stuffing byte loop.
}			End of PES_packet_header
PES_data_payload() {			Start of PES_data_payload.

synchronized_data_packet() {			Start of synchronized_data_packet
data_identifier	8	0x22	Indicates ATSC synchronized data stream.
sub_stream_id	8		This field is privately defined.
PTS_extension_flag	1		1b indicates the presence of a 9-bit PTS extension. 0b indicates that the 9-bit PTS extension is not used.
output_data_rate_flag	1	0b	Always set to 0b.
reserved	2	11b	Reserved fields are always filled with 1b's to avoid start_code emulation.
synchronized_data_packet_ header_length	4		Indicates the number of extra bytes inserted after this field and before the first data payload byte. This 4-bit field shall specify the length of the optional fields in the packet header. This includes the fields that are included when PTS_extension_flag is equal to 1b and it also includes the synchronized_data_private_data_bytes.
if (PTS_extension_flag) {			
reserved	7	1111111b	Reserved fields always contain 1b's to avoid start_code emulation
PTS_extension	9		This field extends the PTS conveyed in the header of this PES packet to allow an extension of the time resolution of synchronized data PTSs from the MPEG-2 standard resolution of 11.1 microseconds (90 kHz) to 37 nanoseconds (27 MHz).
}			End of PTS_extension
for (i=0;i<N1;i++) {			synchronized_data_private_data_bytes
synchronized_data_private_ data_byte	N1* 8		
}			End of synchronized_data_private_data_bytes
remaining data bytes	N * 8		synchronized, streaming payload.
}			End of syncrhonized_data_packet.
}			End of PES_packet_payload.
}			End of PES_packet.

6.3.6 Synchronized Data Streaming of Network Datagrams

The frame structure defined for synchronized data streams is capable of carrying many different data types. One use for this structure is the carriage of datagrams, and more specifically IP datagrams. There are four important facts to remember when encapsulating IP datagrams:

- The use of a LLC/SNAP encapsulation is optional for IP datagrams. However, the ATSC recommends that IP datagrams be carried without using the LLC/SNAP encapsulation. All other network datagrams are required to use the LLC/SNAP encapsulation.
- The Maximum Transfer Unit (MTU) is 4072 bytes when LLC/SNAP is used and 4080 bytes when it is not for all IP datagrams.

- The protocol_encapsulation field attached to the associated tap in the DST should be either 0x07 (LLC/SNAP included) or 0x09 (no LLC/SNAP).
- Only one datagram per PES packet.

If the payload is carrying a datagram, then the first 8 data bytes may contain the LLC/SNAP field. The method to discover if a synchronized data stream carries a LLC/SNAP field is by examining the associated Data Service Table (DST). Entries in the DST indicate the applicable protocol encapsulation via the protocol_encapsulation field. Applicable values for synchronized data streams are listed in Section 6.3.9.

Figure 6.12 illustrates the carriage of an IP datagram in the synchronous data streaming encapsulation.

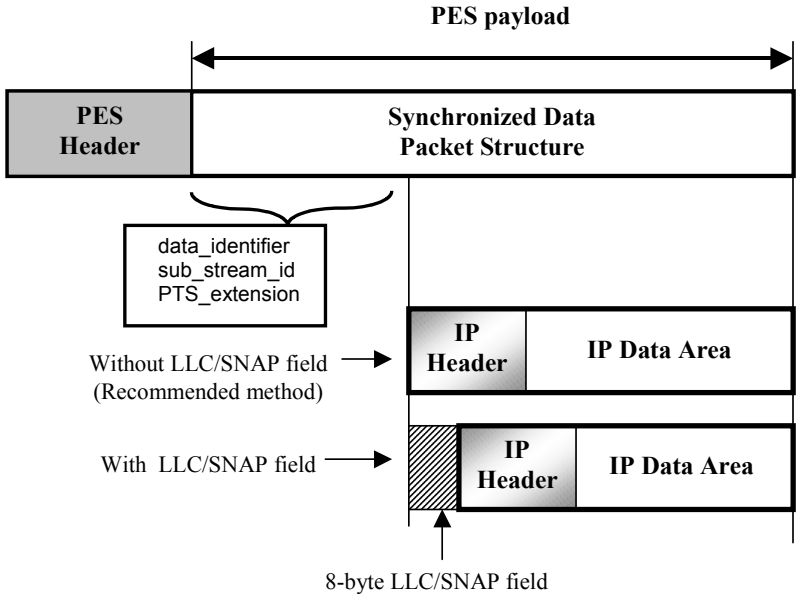


Figure 6.12 Encapsulation of IP packets for synchronized data streaming using PES frames.

6.3.7 PSIP Announcement of Synchronized and Synchronous Data Streams

In PSIP, the Virtual Channel Table defines the list of Virtual Channels applicable for an ATSC Transport multiplex. Each Virtual Channel carries a Service Location Descriptor (SLD) with information connecting the PID and stream_type values. A data receiver may use this information to quickly determine the Program elements. For announcements, every Virtual Channel uses the source_id as a link to program guide information. For data broadcasting, the program announcement information comes in either of two tables: the Event Information Table (EIT) or the Data Event Table (DET).

Synchronous and synchronized data streams may be one of several components of an entire Data Service. If the standalone Data Service requires program guide announcement, it will appear in the Data Event Table (DET). In this case, the Data Service will have a valid data_id value that uniquely identifies its schedule in the program guide. The DET will include a Data Service Descriptor to indicate buffer size and rate constraints required for the service. The DET may also optionally include a PID Count Descriptor that lists the minimum number of

simultaneous PID values that require monitoring. The PID Count Descriptor also optionally provides the total number of PIDs used by the Data Service.

If the Data Service is linked to an audiovisual event (TV program) and does not require a separate announcement, then the Program elements (stream PIDs) for data are treated like the video or audio PIDs of a single event. This event will have an event_id to define its schedule in the Event Information Table (EIT). However, the data_service_descriptor is located in the EIT and indicates the buffer size and the rate constraints required by the Data Service. Likewise, the PID Count Descriptor may optionally appear in the EIT.

6.3.7.1 Data Service Descriptor in PSIP

For any event containing a synchronous or synchronized encapsulation protocol, there must be a data_service_descriptor present in either the Event Information Table (EIT) or the Data Event Table (DET). (Refer to Section 7.1, in this document for additional information on the data_service_descriptor.)

6.3.7.2 PID Count Descriptor

An optional PID_count_descriptor may be included in the DET or EIT of an announced Data Service. (Refer to Section 7.2 in this document for additional information.)

6.3.8 Discovery of Synchronous and Synchronized Data Streaming Program Elements

6.3.8.1 PSIP Service Location Descriptor (SLD) and the Program Map Table (PMT)

PSIP and the MPEG-2 PSI tables (PAT and PMT) are used in data broadcasting for service announcement. The PSIP tables describe in advance the start times and the duration of events so that a data receiver knows when to expect the Data Services to occur. The tables also describe the parental ratings, the Data Service profiles, and similar event features. One particular PSIP table, the Virtual Channel Table (VCT), describes the channel structure in a ATSC Transport multiplex and therefore indicates those channels allocated for Data Services. Both, the VCT and the MPEG-2 PMT indicate how to locate the tables comprising the Service Description Framework (SDF).

The SDF identifies the presence of data being delivered and provides the mechanism for the unambiguous discovery of the data within the current Transport Stream or another Transport Stream using the PSIP and the MPEG-2 PSI (PAT and PMT). (Refer to Section 11 of the ATSC Data Broadcast Standard [1], and Section 7 of this document for additional information regarding Data Service announcements. See Section 12 of the ATSC Data Broadcast Standard [1], and Section 8 of this document for additional information regarding the SDF.)

The Service Location Descriptor (SLD) of the VCT contains the stream_type and elementary_PID information for each synchronous and synchronized protocol Program element of the Data Service. Additionally, the same information is carried in the PMT of the current Transport Stream. The data download protocol stream_type field values associated with the elementary_PID field are defined as follows.

- For the delivery of synchronous streams, the stream_type field of the SLD and the PMT is 0xC2.
- For the delivery of synchronized streams, the stream_type field of the SLD and the PMT is 0x06.

6.3.9 Binding of the Synchronized and Synchronous Data Streaming Program Elements

The Service Description Framework (SDF) provides the binding mechanism for an ATSC Data Broadcast system. The SDF's Data Service Table (DST) must be sent at least once for any ATSC service. The following information must be included as part of the DST. (See Section 8 for additional information regarding the DST.)

Synchronous and synchronized data streams may be one of several Program elements that compose a Data Service. All parameters and configuration of a Data Service are defined in the Data Service Table (DST). The Data Service may be subdivided in one or more applications. Each application may be subsequently subdivided into one or more taps. Linkage from a tap to a particular Program element occurs through the protocol_encapsulation field and the association_tag. The protocol_encapsulation field identifies the encapsulation protocol used by the associated Program element. Table 6.18 lists the possible values for synchronous and synchronized Program elements.

Table 6.18 Synchronous and Synchronized Protocol_Encapsulation Field Values

Encapsulation Protocol	Value of the protocol_encapsulation Field
Synchronized data streams encapsulated in PES	0x05
Synchronous data streams encapsulated in PES	0x06
Synchronized data streams using LLC/SNAP for multiprotocol support	0x07
Synchronous data streams using LLC/SNAP for multiprotocol support.	0x08
Synchronized IP data streams using PES	0x09
Synchronous IP data streams using PES	0x0A

Once the DST is located, a data receiver retrieves the tap information including the protocol_encapsulation field. A final binding is necessary between a tap in the DST and an elementary stream in the PMT. This final binding is accomplished by using the association_tag. There is an association_tag for each of the taps that appear in a DST and there is an association_tag_descriptor linked to the corresponding elementary stream in a PMT.

6.4 Data Piping

The Data Piping encapsulation protocol is specified in Section 10 of the ATSC Data Broadcast Standard [1].

Data Piping is an asynchronous transportation mechanism for data. Data is inserted directly in the payload of MPEG-2 Transport Stream packets.

There is no defined mechanism for splitting and reassembly of the datagrams. 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.

The continuity_counter will be used as defined by MPEG-2 ([2], Section 2.4.3).

6.4.1 PSIP Announcement of the Data Piping Protocol

In PSIP, the Virtual Channel Table defines the list of Virtual Channels applicable for an ATSC Transport multiplex. Every Virtual Channel carries a Service Location Descriptor (SLD) with information connecting the PID and stream_type values. A data receiver may use this information to quickly determine the Program elements. For announcements, every Virtual Channel uses the source_id as a link to program guide information. For data broadcasting, the program

announcement information comes in either of two tables: the Event Information Table (EIT) or the Data Event Table (DET). A Data Service containing a Data Piping encapsulation protocol Program element may be announced in either an EIT or a DET. (For further information regarding Data Service announcement specifics, see Section 7 of this document.)

6.4.1.1 Data Service Descriptor in PSIP

If the Data Piping encapsulation protocol is an announced event then there must be a data_service_descriptor present in either the Event Information Table (EIT) or the Data Event Table (DET). (Refer to Section 7.1, in this document for further information on the data_service_descriptor.)

6.4.1.2 PID Count Descriptor

An optional PID_count_descriptor may be included in the DET or EIT of an announced Data Service. (Refer to Section 7.2 in this document for additional information.)

6.4.2 Discovery of a Data Piping Protocol Program Element

6.4.2.1 PSIP Service Location Descriptor (SLD) and Program Map Table (PMT)

PSIP and the MPEG-2 PSI tables (PAT and PMT) are used in data broadcasting for service announcement. The PSIP tables describe in advance the start times and the duration of events so that a data receiver knows when to expect the Data Services to occur. The tables also describe the parental ratings, the Data Service profiles, and similar event features. One particular PSIP table, the Virtual Channel Table (VCT), describes the channel structure in a ATSC Transport multiplex and therefore indicates those channels allocated for Data Services. Both, the VCT and the MPEG-2 PMT indicate how to locate the tables comprising the Service Description Framework (SDF).

The SDF identifies the presence of data being delivered and provides the mechanism for the unambiguous discovery of Data Piping data within the current Transport Stream or another Transport Stream using PSIP and the MPEG-2 PSI (PAT and PMT). (Refer to Section 11 of the ATSC Data Broadcast Standard [1], and Section 7 of this document for additional information regarding Data Service announcements. See Section 12 of the ATSC Data Broadcast Standard [1], and Section 8 of this document for additional information regarding the SDF.)

The Service Location Descriptor (SLD) of the VCT contains the stream_type and elementary_PID information for each Data Piping encapsulation protocol Program element of the Data Service. Additionally, the same information is carried in the PMT of the current Transport Stream. stream_type field values associated with the elementary_PID field are defined as follows:

> The stream_type field of the SLD and the PMT may carry any value. (Note that A/90 specifies that data piping shall not be used for the carriage of video elementary streams (stream type 0x02) or audio elementary streams (stream type 0x81). Other ATSC standards define transmission of these two stream types.

The data delivery PID for the Data Piping encapsulation protocol will be specified by the elementary_PID of the SLD and the PMT. The declared stream_type associated with the elementrary_PID must match the required stream_type value.

6.4.3 Binding of a Data Piping Protocol Program Element

The Service Description Framework (SDF) provides for the binding mechanism for an ATSC data broadcast system. The Data Service Table (DST) must be sent at least once for any ATSC service. The following information must be included as part of the DST. (See Section 8 for additional information regarding the DST.)

6.4.3.1 Protocol Encapsulation

The protocol_encapsulation value is located in the Data Service Table (DST). The protocol_encapsulation field is set to 0x0B for the delivery of Data Piping encapsulation protocol.

7. PSIP DATA SERVICE EVENT ANNOUNCEMENT

A Data Service is announced by an event in either the PSIP Event Information Tables (EIT) (defined in [6]) or the newly defined Data Event Tables (DET) (defined in [1]).

Like an EIT-k table, a DET-k table consists of multiple table instances where each instance is identified by the value of the source_id field in the table section header. An instance of a DET-k is therefore associated with a given ATSC Virtual Channel. The similarity between DET-k's and EIT-k's does not stop here. Like any EIT-k, a DET-k is associated with a given 3-hour time window. The syntax of a DET-k instance is similar to the syntax of an EIT-k instance except for the fact that there is a data_id field and not an event_id field is used to identify each event. The data_id field serves the same purpose as the event_id field which is to facilitate the association of Extended Text data with an event. The value space for the data_id fields is independent of the value space for the event_id fields, meaning that a data_id and event_id can have the same value for two distinct events. The DET-k's have their own table_type value in the PSIP Master Guide Table. The range of values is 0x1100-0x117F. Likewise, the ETT's associated with the DET Tables are in the range 0x1200-0x127F.

The choice of EIT-k's or DET-k's to announce the schedule of a data service is driven by the consideration of the following scenarios:

- First Scenario: The data service belongs to a data-only ATSC Virtual Channel. The minor_channel_number associated with such channel must be equal or greater than 100 and the service_type of the channel is equal to 0x04. In this case, the title_text structure of the DET-k instances announcing the schedule of the data services specifies the title of the data service. The data service has its own schedule and its own rating.
- Second Scenario: The data service is an enhancement to an ATSC audio (service_type 0x03) or audiovisual (service_type 0x02) Virtual Channel. The data service portion of the event and the audiovisual portion of the event share the same schedule, meaning that the data service is reproduced and available to the viewer during the time scheduled for the audiovisual event. Note that this does not mean that data service data is continuously transmitted during the integral scheduled period of the audiovisual event. For example, a data service may only appear during a commercial break. The data service also shares the same name as the audiovisual portion of the event. The PSIP title_text structure may capture or may not capture language or character codes revealing the presence of a data service. In this case, the data service schedule is announced by means of EIT-k instances. Consequently, these instances announce the schedule of an audio-visual-data event in an ATSC Virtual Channel.
- Third Scenario: The data service is a data event associated with an audio or audiovisual channel. In this case, the schedule of the data event may not coincide with the schedule of the audiovisual events in the virtual channel. For example, the schedule of the data event

may be such that it overlaps with the second half of an audiovisual event. Likewise, the schedule of the data event may be covering only the first quarter of an audiovisual event. Furthermore, in this case, the name of the data service may be different from the name of the audiovisual event with which it is running concurrently in the same ATSC Virtual Channel. In this case, DET-k instances are used to convey the schedule and the title of the data service portion of the event. The DET-k instances and the associated EIT-k instances convey the same source_id value in their header bytes as they refer to the same ATSC Virtual Channel. The service_type of the ATSC Virtual Channel is 0x02 or 0x03.

A third table (the Long Term Service Table (LTST)) is defined in A/90 to pre-announce data events that will occur on a timescale longer than is available through the EIT/DET mechanism. Discussion of the LTST is omitted in this document for simplicity.

Virtual channels that carry data services may be hidden from navigation and/or program guides. This function is supported in the VCT using two bits called hidden and hide_guide respectively. When these two bits are each set to '1' then the channel is hidden from navigation and the EPG. When the hidden bit is '1' and the hide_guide bit is '0', the channel is hidden from navigation but may appear in an EPG. All other combinations are ignored.

The DET is constructed and identified in the same manner as the EIT except for a different table_id. The DET table_id is 0xCE. Refer to section 6.5 of PSIP [6], and Annex D of [6] for further information regarding the construction of the DET/EIT. The event_id and data_id fields do not share the same value space.

The recommended cycle time for DET-0 is 500 milliseconds. The recommended cycle time for DET-1 is 2 seconds. In all cases, it is recommended that the maximum cycle time for a DET-k be such that it is always smaller or equal to the DET of the next time window, DET-(K+1), if it exists.

Figure 7.1 illustrates the DETs relation to the EIT.

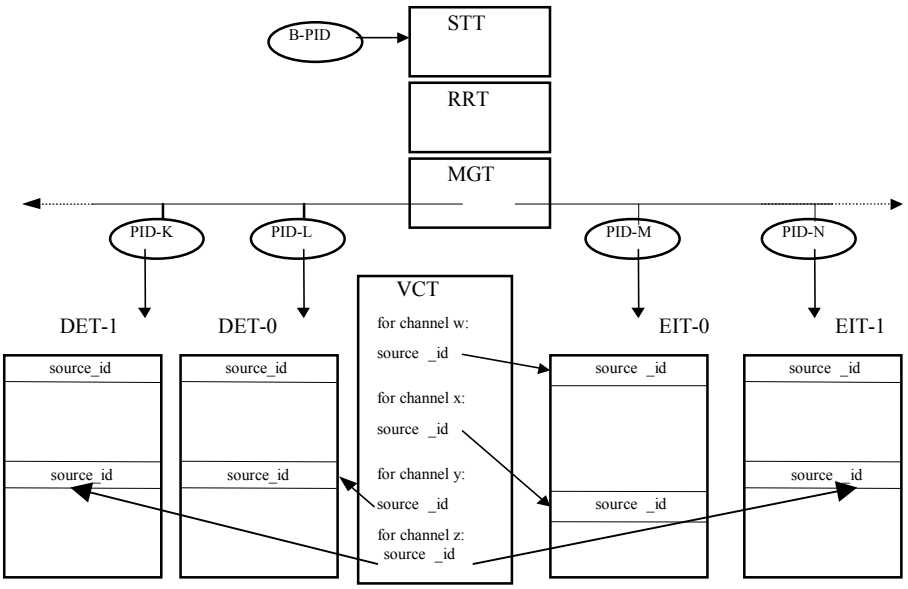


Figure 7.1 Event Table hierarchy.

7.1 Data Service Descriptor

Every event in the DET must have a data_service_descriptor. If the Data Service is announced in an EIT, then each event with an associated Data Service will also carry a data_service_descriptor. The purpose of the data_service_descriptor is to publish the Data Service profile and level.

The data_service_profile specifies the maximum data bit-rate that will be transmitted with this Data Service. The maximum data rate is inclusive of the Transport Stream header fields as well as the Data Service discovery information such as that carried in the Data Service Table and Network Resource Table. The maximum data rate is applied to all the data elementary streams comprising the Data Service. (Additionally, see Section 11.5 of the ATSC Data Broadcast Standard [1].) The data_service_profile values 0x01, 0x02 and 0x03 signal data services to which is guaranteed a certain and fixed bandwidth. The data_service_profile value 0x01 means that the guaranteed bandwidth assigned to the data service is 383896 bits/sec. This means that insertion of data into the ATSC multiplex can be sustained at that maximum rate at any moment during the data service transmission. The value 0x02 indicates that the guaranteed rate is 3838960 bits/sec and the value 0x03 indicates that the guaranteed rate is 19.2 Mbits/sec. The data_service_profile value 0x4 indicates that the data service is opportunistic in nature meaning that the instantaneous rate available for insertion of the data into the ATSC multiplex varies in time. Reasons for such variations of bandwidth in time may come for large swing in the number of bits taken at any instant by the transmission of the video and/or audio elementary streams. The instantaneous bandwidth of an opportunistic data service may vary from 0 bits/sec to 19.2 Mbits/sec. However, a data service for which the instantaneous bandwidth varies for example, between 2 Mbits/sec and 19.2 Mbits/sec should also be classified as an opportunistic data service. In this case, although the service may be considered as the superposition of a guaranteed data service of 2 Mbits/sec and an opportunistic data service, it must be observed that the overall data service is indeed opportunistic since it is very clear that the instantaneous bandwidth available at the head-end to transmit the data is variable in time. In general, any data service for which the instantaneous bandwidth varies between some fixed floor value all the way to 19.2 Mbps per second should be signaled as opportunistic (value 0x04).

The data_service_level is tied to the specification of a Data Elementary Stream buffer size. Upon parsing the data_service_descriptor, a data receiver can determine whether the buffer requirements for acquiring the Data Service match its buffer and throughput capability.

The data_service_profile along with the data_service_level can be used to determine the maximum latency, sb_leak, and sb_size parameters associated with all the data elementary streams comprising the Data Service. (For further discussion regarding buffering requirements, see Section 9 of this document.)

Table 7.1 describes the data_service_descriptor.

Table 7.1 Data Service Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
data_service_descriptor() {			
descriptor_tag	8	0xA4	
descriptor_length	8	0x03 or greater	0x03 if only the data_service_profile and data_service_level are specified. 0x04 or greater if there is private_data_byte(s) in the descriptor.
data_service_profile	8		
data_service_level	8		
private_data_length	8		0x00 if the descriptor_length is 0x03.
for (i=0; i<private_data_length; i++) {			
private_data_byte	8		
}			
}			

7.2 PID Count Descriptor

The descriptor loop of either the DET or the EIT may optionally include a PID_count_descriptor providing the total number of PIDs used by the Data Service as well as the minimum number of PIDs used in the Data Service, which will provide a reasonable rendering of a subset of the service. A data receiver may use the number of PIDs to determine whether it has the PID filtering capabilities available to successfully receive the Data Service.

Table 7.2 describes the PID_count_descriptor.

Table 7.2 PID Count Descriptor Syntax

Field Name	No. of Bits	Field Value	Notes
PID_count_descriptor() {			
descriptor_tag	8	0xA5	
descriptor_length	8	0x02 or 0x04	0x02 if only total_number_of_PID is specified. 0x04 if both total_number_of_PID and min_number_of_PID are specified.
reserved	3	111b	
total_number_of_PIDs	13		
reserved	3	111b	
min_number_of_PIDs	13		
}			

8. DATA SERVICE PROGRAM ELEMENT DISCOVERY AND BINDING

The ATSC Service Description Framework (SDF) enables a data receiver to discover the data services carried in an ATSC transport multiplex and bind to the data in the data services. It enables the receiver to find out what data are in each data service, where the data are located,

what encapsulation protocol or protocols are used to transmit them, and certain other related information.

A data service in an ATSC transport multiplex must be contained in a virtual channel, and each virtual channel may have at most one data service. A virtual channel containing a data service may be a video channel (service_type 0x02 in the Virtual Channel Table), an audio channel (service_type 0x03) or a data-only channel (service_type 0x04). A single data service may consist of multiple *applications*, and each application may contain multiple data elements.

Each data element for a data service is listed in the MPEG-2 Program Map Table (PMT) for the MPEG-2 program corresponding to the virtual channel, and in the Service Location Descriptor (SLD) for the virtual channel in the Virtual Channel Table (VCT). In the descriptor loop for each data element in the PMT there must be an *association tag* descriptor, which provides a 16-bit identifier (association tag) for the data element. These association tags must be unique within the scope of the virtual channel (MPEG-2 program). The association tag for a data element remains invariant under demultiplexing and remultiplexing of the virtual channels in a transport stream (unlike the packet identifier (PID) associated with a program element, which may change during such demultiplexing/remultiplexing operations). The detailed syntax of the association tag descriptor is described in Section 8.1 of this document.

Note that there should be a one-to-one correspondence between the data elements listed in the SLD for the virtual channel and the data elements listed in the PMT for the MPEG-2 program corresponding to the virtual channel.

Two tables are used to convey the discovery and binding information for a data service, the Data Service Table (DST) and the Network Resources Table (NRT). Both of these tables are carried in a single MPEG-2 program element in the virtual channel containing the data service. This program element is identified by having stream_type 0x95 in the virtual channel's PMT and VCT SLD. The DST is required for all virtual channels containing a data service, but the NRT is only required when the data service requires data that is not contained in the virtual channel itself (such as data on a remote server accessed over an external network connection, or data in some other virtual channel in the broadcast stream).

Since there may be at most one data service in a single virtual channel, there may be at most one program element in a virtual channel with stream_type 0x95. Moreover, different virtual channels must use different program elements for their DSTs and NRTs. It is not allowed to have DSTs or NRTs for different data services in the same program element.

Thus, when a data receiver is tuned to a virtual channel containing a data service, it first acquires the virtual channel's PMT and/or VCT SLD and finds in it the program element with stream_type 0x95. It then looks in that program element and acquires the DST and, if present, the NRT.

The DST contains the following information:

- Protocol version for the SDF itself (0x01 for the current version of A/90).
- List of the applications in the data service.
- For each application:
    1) Compatibility descriptor (describing capabilities needed to render the application).
    2) An application identifier structure (app_id_byte_length and app_id_byte fields).
    3) List of the resources which make up the application.
    4) For each resource:

a) protocol_encapsulation field giving encapsulation protocol used for the resource.
b) action_type flag indicating how the resource is to be used.
c) resource_location flag indicating whether the resource is local or remote.
d) Unique tap_id identifier for the resource (scoped by the application).
e) association_tag and an optional *selector* used to locate the resource.
f) List of descriptors for the resource.

5) List of descriptors for the application.
6) Private data area for the application (containing initialization data, or whatever).

- List of descriptors for the data service.
- Private data area for the data service (containing initialization data, or whatever).

The protocol_encapsulation field tells which of the ATSC Data Broadcast encapsulation protocols is used to carry the resource.

The action_type flag indicates whether or not the resource should be auto-launched in order to start the application.

The tap_id is used within the application to reference the resource.

If the resource_location flag indicates the resource is local, then the receiver matches the association tag for the resource in the DST with an association tag in the PMT to identify the data element containing the resource. If the resource_location flag indicates the resource is remote, then the receiver matches the association tag for the resource in the DST with an association tag in the NRT to identify where to find the resource.

The selector field is used to reference a subset of the resource identified by the association tag. For example, the association tag may identify an entire data carousel, and the selector field may be used to specify a single module of the carousel, or the association tag may identify a data element containing IP packets, and the selector field may be used to specify that the resource of interest consists of only those packets with a particular IP destination address.

A detailed description of the DST can be found in Section 8.4 of this document.

The NRT contains the following information:

- List of the remote resources used in the data service.
- For each remote resource:
  1) Compatibility descriptor (describing capabilities needed to access the resource).
  2) DSM-CC resource descriptor, containing the following information:
     a) Resource descriptor type (identifying the type of resource).
     b) Resource num identifier for the resource (unique within the NRT).
     c) Association tag (unique within NRT, used by the DST to reference the resource).
     d) Resource data fields used to completely describe the resource, with format dependent on resource descriptor type.
     e) A few other fields that are unused (applicable only in other DSM-CC contexts) private data for the resource (containing initialization data, or whatever)

The DSM-CC resource descriptors specifically listed in the ATSC Data Broadcast Standard are:

- Deferred MPEG Program Element descriptor, used to identify an MPEG-2 resource in another virtual channel (in the same or a different transport stream).
- IP Resource descriptor, used to identify a resource by IP address and port.
- IPv6 Resource descriptor, used to identify a resource by IPv6 address and port.
- URL Resource descriptor, used to identify a resource by its URL.

A detailed description of the NRT can be found in Section 8.5 of this document.

Note that if multiple applications are merged to form a data service, their local data elements are merged into a single MPEG-2 program, and their remote resources are merged into a single NRT. In this situation care must be taken to avoid collisions of association tags in the PMT and in the NRT.

The DST must be transmitted at least once during the scheduled delivery of the data service (i.e., between the start time and end time of the EIT or DET event containing the data service). Since a receiver typically cannot begin rendering the data service until it has seen the DST, it is a good idea to transmit it shortly before transmitting any data and periodically after that. The shorter the interval between transmissions of the DST, the faster the receiver will be able to start rendering the data service when a viewer tunes to the virtual channel in mid-event. The frequency of transmission of the DST represents a compromise between the bandwidth required for it and the latency experienced by a viewer tuning to the channel in mid-event. Similar considerations apply to the NRT, when remote data is part of the data service.

Figure 8.1 illustrates the relationships among the MPEG-2 PMT, the ATSC PSIP VCT, and the ATSC Data Broadcast DST and NRT.

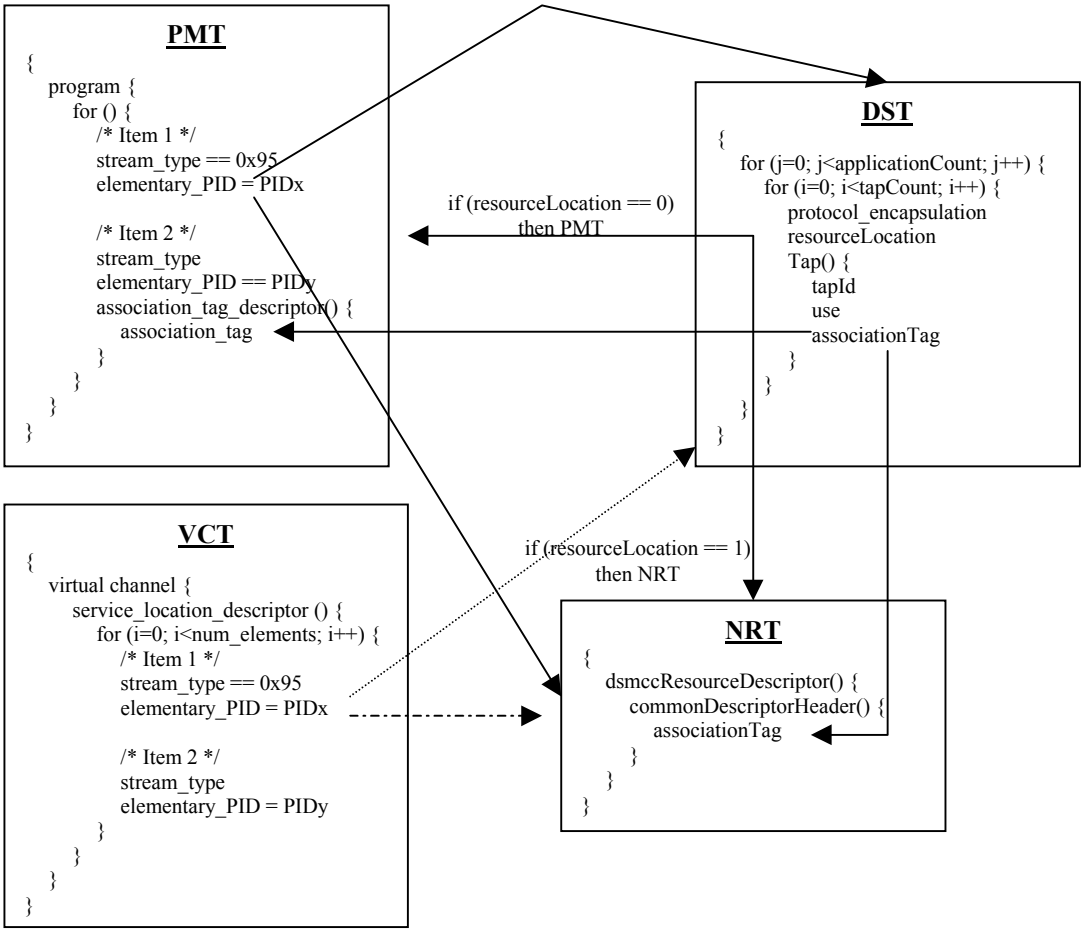


Figure 8.1 The Relationship of the VCT, PMT, DST, and NRT in the SDF

In Figure 8.1 the MPEG-2 Transport Stream packets conveying the DST and the NRT have the same elementary_PID value, denoted by PIDx. That elementary_PID value is referenced by both the PMT and the SLD in the VCT, using stream_type 0x95 to identify it. The DST and the NRT are conveyed in distinct tables within that program element to provide a de-coupling, allowing each table to be transmitted independently and at a different frequency. The receiver can easily tell the tables apart, because they have different table_id values, 0xCF for the DST and 0xD1 for the NRT. Each association tag value in the DST references either an association_tag descriptor in the PMT (for a local resource) or an association tag in one of the DSM-CC resource descriptors in the NRT (for a remote resource).

Note that there are no association tag descriptors in the SLD, since there is no descriptor loop for the program elements in the SLD where they could appear. Thus, the binding to the local data elements must always go through the PMT, not the VCT. This means that when a viewer tunes to a virtual channel containing a data service, the receiver must acquire the VCT so it can determine what MPEG-2 program corresponds to the virtual channel number requested by the viewer, and it must then acquire the PMT for that program in order to bind to the data elements.

Any association tag that appears in the DST must have a corresponding association tag in the PMT or in the NRT, depending on whether the resource for the association tag is identified by

the resource_location flag as local or remote. However, there may be association tags in the PMT or the NRT for which there is no corresponding association tag in the DST (i.e., there may be local or remote resources that are identified but not used by the data service).

The delivery of the SDF information (DST and NRT) follows the buffer model for asynchronous data services. (See Section 9 for additional information on buffer models.)

8.1 Association Tag Descriptor

Table 8.1 describes the Association Tag Descriptor that is used to bind resources in the DST to program elements in the PMT. One of these descriptors goes in the ES_info descriptor loop of the PMT, for each data element in the virtual channel.

Table 8.1 Association Tag Descriptor

Field Name	No. of Bits	Field Value	Notes
association_tag_descriptor() {			
descriptor_tag	8	0x14	denotes association tag descriptor
descriptor_length	8	0x05	length in bytes of this descriptor, after this field
association_tag	16	arbitrary	association tag value; must be unique within the PMT containing this descriptor
use	16	0x1000	denotes “not applicable”
selector_length	8	0x00	no selector field allowed here
}			

The generic syntax of this descriptor, as defined by the DSM-CC standard, allows for private data bytes right after the selector field, but this is prohibited by the ATSC Data Broadcast Standard.

The reason that no selector field is allowed in the association tag descriptor is that it is not needed, since a selector field may be present in any DST entry that references the association tag.

8.2 Data Service Discovery–The Basic Algorithm

Data Service discovery relies on the usage of the structures identified in the following sections. The acquisition from the ATSC Transport multiplex may or may not be concurrent.

8.2.1 PSIP Structures

The schedule of the Data Service is discovered though the PSIP EIT-k’s or DET-k’s. The PSIP MGT is acquired to identify the elementary_PID value of the Program elements conveying the EIT-k’s and DET-k’s. The existence of a Data Service in a PSIP Virtual Channel can be detected by the presence of a data_service_descriptor in the descriptor loop associated with the event(s) in which the Data Service appears. This fact is true for standalone Data Services and Data Services related to an audio-visual event (service_type 0x04 and 0x02/0x03, respectively). It is also true whether the schedule of the Data Service events appear in EIT-k’s or DET-k’s.

The PSIP VCT is acquired and reconstructed in the data receiver. In the VCT, the Service Location Descriptor (SLD) of the Virtual Channel containing the Data Service is used to identify the elementary_PID value of the MPEG-2 Transport Stream packets conveying the A/90 Service

Description Framework information (DST and NRT). The SLD is parsed to identify the Program elements of the MPEG Program that are used in the Data Service. The associationTag values associated with these Program elements are validated as references to data elementary streams in the Data Service. There may be other associationTag values within the same MPEG-2 program that do not refer to Program elements belonging to the PSIP Virtual Channel containing the Data Service of interest and therefore can be disregarded.

8.2.2 MPEG-2 Structures

The Program Association Table (PAT) bytes are acquired from Transport Stream packets with the PID value equal to 0 and the table is reconstructed in the data receiver. The PAT identifies the program_map_PID value of the Transport Stream packets conveying the Program Map Table (PMT) bytes for a particular MPEG-2 Program (identified by the program_number). The elementary_PID and stream_type values of the Program elements belonging to the MPEG-2 program associated with the PSIP Virtual Channel are discovered. During this discovery process, the correspondences between an elementary_PID value and an associationTag value in an association_tag_descriptor structure are recorded.

8.2.3 A/90 Service Description Framework Information

The Data Service Table (DST) bytes are acquired following identification of the Program element of stream_type 0x95. A list of valid associationTag is used to resolve the location of Program elements abstracted by the Tap to the MPEG-2 Program elements in the ATSC Transport multiplex. The Network Resources Table is acquired, if needed, to discover the other communication channels that are also needed by the Data Service. Communication channel setup, management and termination is handled in the data receiver depending on the type of resource descriptor used to announce the communication channel. (The communication channel is outside of the scope of the ATSC Data Broadcast Standard [1] and this document.)

8.3 Data Service Table (DST)

The Data Service Table (DST) provides for the discovery of a Data Service and the binding of an application to it's applications resources. Application discovery and binding is achieved using the following five sets of data structures:

- A DSM-CC compatibility descriptor (compatibility_descriptor structure) to signal compatibility and requirements of an application. The compatibility requirement can be software and/or hardware requirements.
- An application identifier structure (app_id_byte_length and app_id_byte fields) to convey the name of the application.
- A list of DSM-CC Tap structures and companion fields.
- An application data structure (app_data_length and app_data_byte fields) to convey application input parameters.
- A list of application specific descriptors for conveying information about the application (app_info_length and descriptors).

A Data Service declaration also includes a service information structure to convey service specific descriptors (serviceInfoLength field and descriptor() fields). In addition, private service information can be appended to the end of the Data Service Table structure (servicePrivateDataLength field and servicePrivateDataByte fields).

The DST may span more than one data_service_table_section. DST transmission is accomplished by partitioning the table into a series of sections. The last_section_number field in the MPEG-2 Private Section header specifies the number of sections used. It is recommended that the splitting of a DST into multiple sections be such that information associated with each application of the Data Service is confined to a single section.

At least one instance of the DST must be transmitted during the PSIP schedule of a Data Service. While it is possible to construct a null DST by setting both the application_count_in_section and service_info_length fields to zero, this contravenes the intended purpose of the DST[4]. As described previously, the DST is used to bind an application to its resources, meaning that the DST carries the information needed to find the data used by an application. Without the capability for dynamic binding, data services can only be consumed by proprietary receiver implementations, losing the possibility of interoperability between generic data services and receivers.

The data_services_table_sections will be carried in MPEG-2 Transport Stream packets similar to Section 6.1.8 except the DSMCC_section will be replaced by data_services_table_sections. Refer to Section 6.1.8 for associated information.

The following data structure describes the DST encapsulated in a data_services_table_section (see Table 8.2).

Table 8.2 Data Services Table

Field Name	No. of Bits	Field Value	Notes
data_services_table_section() {			
table_id	8	0xCF	
section_syntax_indicator	1	1b	
private_indicator	1	1b	
reserved	2	11b	
private_section_length	12		The length in bytes starting from table_id_extension field to the end of the data_services_table_section.
table_id_extension	16	0xFFFF	ATSC reserved.
reserved	2	11b	
version_number	5		
current_next_indicator	1	1b	
section_number	8		
last_section_number	8		
data_service_table_bytes() {			Data Service Table
sdf_protocol_version	8		This value is initially set to 0x01. The other values are ATSC reserved so in the future, additional SDF protocols can be signaled.

4 This topic was the subject of significant debate during the drafting process. Many participants felt that a null DST should have been disallowed by the standard. In practice, filling out the fields in the DST with meaningful values will enhance a receiver's capability to deal with data services and is strongly encouraged.

application_count_in_section	8		Number of applications described by this DST section.
if(application_count_in_section > 0){			
for (j=0; j<application_count_in_section; ++) {			Beginning of the data for each application.
compatibility_descriptor() {			Signal application requirements.
compatibilityDescriptorLength	16		Length in Bytes is from descriptorCount to the end of the compatibility_descriptor.
descriptorCount	16		
for (i=0; i<descriptorCount; i++){			
descriptorType	8		
descriptorLength	8		
specifierType	8		
specifierData	24		
model	16		
version	16		
subDescriptorCount	8		
for(m=0; m<subDescriptorCount; m++) {			
subDescriptor() {			
subDescriptorType	8		
subDescriptorLength	8		
for(k=0; k<subDescriptorLength; k++){			
additionalInfo	8		User private data.
}			
}			
}			
}			
}			End compatibility_descriptor.
app_id_byte_length	16		Length of application identifier.
if(app_id_byte_length > 1){			
app_id_description	16		Specifies the format and the semantics of the following application identifier bytes.
for (i=0; i<app_id_byte_length-2; i++) {			
app_id_byte	8		Application Identifier.
}			
}			
tap_count	8		Number of Tap structures plus other information for each Tap.
for (i=0; i<tap_count; i++) {			
protocol_encapsulation	8		Specifies the encapsulation mechanism used to convey the data element referred to by the Tap.

action_type	7		Nature of the data referenced by the Tap. 0x00 = Run-time data. 0x01 = bootstrap data. Values 0x02 to 0x3F are ATSC reserved.
resource_location	1		Location of the resource description. 0b = PMT, 1b = DSM-CC Resource Descriptor in the NRT. The value of this field shall not be equal to '0' when the value of the protocol_encapsulation field is equal to 0x00, since a value of 0x00 signals that the data will be coming from something other than an MPEG-2 Transport Stream.
Tap() {			See Section 7.5.1 below for additional information.
tap_id	16		Unique identifier of the Tap within the application.
use	16	0x0000	Not used. The value 0x0000 indicates that existence and format of such additional protocols is unknown. This is the only recognized value at this point.
association_tag	16		The value corresponds to the value of the association_tag field in either the Association Tag Descriptor located in a PMT or in the commonDescriptorHeader structure of a dsmccResourceDescriptors residing in the NRT. The association_tag's location is specified by the resource_location field.
selector() {			Further specifies the data element relative to the "base" defined by the associationTag.
selector_length	8		Length of the selector structure. Permitted values depend on the value of the protocol encapsulation value associated with this selector.
if (selector_length>0) {			
selector_type	16		This field specifies the format of the selector bytes. Defined values are in the range 0x0101-0x0108. Not all values are permitted given a protocol encapsulation value. See the ATSC Data Broadcast Standard [1] Table 12.9 for extensive information about the meaning of each value.
for (m=2; m<selector_length; m++) {			
selector_byte	8		
}			
}			
}			
}			End of the Tap structure.
tap_info_length	16		

for (k=0; k<N; k++) {			
descriptor()			Tap level descriptors.
}			
}			End Tap loop.
app_info_length	16		
for (i=0; i<M; i++) {			
descriptor()			Application level descriptors.
}			
app_data_length	16		
for (i=0; i<app_data_length; i++) {			
app_data_byte	8		Application input parameters.
}			
}			
service_info_length	16		
for (j=0; j<K; j++) {			
descriptor()			Service level descriptors.
}			
service_private_data_length	16		
for(j=0; j<service_private_data_length; j++){			Service level private data.
service_private_data_byte	8		
}			
}			End of if statement on application_count_in_section > 0
}			end of application loop in section
CRC_32	32		
}			

The protocol_encapsulation field indicates the MPEG-2 packetization, synchronization, error detection and the encapsulation protocol used to convey a data element pointed to by the Tap. A value of 0x00 for the protocol_encapsulation field indicates that the data element will be obtained from another source than an MPEG-2 Transport Stream (for example, an IP resource). For this reason, the resource_location field indicates that the location of the resource description is in the DSM-CC Resource Descriptor in the NRT, rather than in the PMT[5].

8.3.1 Tap Structures

A Tap is a fundamental structure, defined in [4], that allows binding of data components consumed by an application with their location in a logical communication channel. A Tap structure is used for referencing the location of a particular application-level data component (like a file, IP datagram stream, or a DSM-CC data module) in a specific logical connection (like an IP virtual channel or an MPEG-2 Program element).

A Tap includes the following fields:

5 Note: Aside from the compatibilityDescriptor, all the descriptors mentioned elsewhere in this document are of the form of MPEG-2 Systems descriptor (meaning a structure starting with a descriptor_tag and descriptor_length 8-bit fields). The DSM-CC Resource Descriptors described here and in Section 12.3.2 of [1] are not MPEG-2 Systems descriptors and can only go in an NRT.

- A tap_id field that can be used by content developers as a handle to a data component consumed by the application. The content provider chooses the tap_id values. The content provider must ensure at authoring time that tap_id values related to an application are unique. Note that the tap_id values are scoped by the application and therefore, there is no need to manage them at the service level.
- The association_tag field abstracts the logical communication field. It is used at binding time to refer to either an association_tag field in an association_tag_descriptor in the MPEG-2 Program Map Table of the Data Service or to an associationTag field in the commonDescriptorHeader of a DSM-CC resource descriptor in the Network Resources Table of the Data Service. The same association_tag value may be used in more than one Tap.
- The selector byte fields specify the data component conveyed in the logical communication channel. The first byte in the selector is always a selector_type field specifying the format and semantics of the selector bytes to follow.

There are multiple supported selectorType formats. The following is an overview of several of the selectors.

- The selector_type value 0x0101 signals that the selector_byte fields indicate a moduleId value of a data module. This selector_type value may only be used under protocol_encapsulation values 0x01, 0x02 and 0x0D. In this case, the value of selector_length is equal to 4.
- The selector_type value 0x0102 signals that the selector_byte fields indicate a valid deviceId address. This selector_type value may only be used under protocol_encapsulation values 0x03 and 0x04. The value of the selector_length field must be in agreement with the number of active devideId bytes specified by the multiprotocol_encapsulation_descriptor structure. In the devideId field, from 1 to 6 bytes can be active.
- The selector_type value 0x0103 signals that the selector_byte fields indicate a PES sub-stream identifier. This selector_type value may only be used under protocol_encapsulation value 0x05. In this case, the value of selector_length is equal to 0x03.
- The selector_type value 0x0104 signals that the selector_byte fields indicate a destination Network Address. The destination addresses are conveyed in the SNAP header of the datagrams. This value of selector_type may only be used under the protocol_encapsulation values 0x03, 0x07 and 0x08. In this case the value of selector_length is variable. .
- The selector_type value 0x0105 signals that the selector_byte fields indicate a destination IPv4 (Internet Protocol Version 4) address. This value of selector_type may only be used under the protocol_encapsulation values 0x04, 0x09 and 0x0A. In this case, the value of selector_length is equal to 6.
- The selector_type value 0x0106 signals that the selector_byte fields indicate a destination IPv6 (Internet Protocol Version 6) address. This value of selector_type may only be used under the protocol_encapsulation value 0x04, 0x09 and 0x0A. In this case, the value of selector_length is equal to 18.
- The selector_type value 0x0107 signals that the selector_byte fields indicate a groupId field of a two-layer control data carousel. This value of selector_type may only be used under the protocol_encapsulation value 0x01, 0x02 and 0x0D. In this case, the value of selector_length is equal to 6.
- The selector_type value 0x0108 signals that the selector_byte fields indicate a groupId and a moduleId field of a two-layer control data carousel. This value of selector_type may only

be used under the protocol_encapsulation value 0x01, 0x02 and 0x0D. In this case, the value of selector_length is equal to 6.

The resource_location field that precedes the Tap in the Data Service Table is used to facilitate the binding of a Tap to a particular logical connection. Figure 8.1 illustrates the use of a Tap by an application. The Tap provides a mapping between data consumed by an application and location of this data on a particular logical connection.

8.4 Network Resource Table (NRT)

The Network Resources Table (NRT) provides a list of all interactive and external broadcast logical connections used by the Data Service. The NRT provides the location of those resources not found in the current MPEG-2 program or MPEG-2 Transport Stream. (The current Transport Stream resources are described in the PSIP Service Location Descriptor and the MPEG-2 Program Map Table.) Each of these connections is described by a Resource Descriptor structure as defined in DSM-CC [4]. While DSM-CC [4] has described a number of Resource Descriptors, only the following have been defined in the ATSC Data Broadcast Standard [1] and can be expected to be transmitted:

- deferredMpegProgramElement Resource Descriptor (resourceDescriptorType = 0x0014)
- Internet Protocol Version 4 Resource Descriptor (resourceDescriptorType = 0x0009)
- Internet Protocol Version 6 Resource Descriptor (resourceDescriptorType = 0x0015)
- URL Resource Descriptor (resourceDescriptorType = 0x0016)

In particular, each Resource Descriptor features an associationTag value in its commonDescriptorHeader providing a unique identification of the logical connection. This associationTag matches one or several association_tag values found in the various Taps of a Data Service Table.

The association_tag values are scoped by an MPEG-2 program number. This means that the deferredMPEGResourceDescriptor is needed to reference any Program Element that does not belong to the current MPEG-2 program (or equivalently, to the same virtual channel).

The NRT information may span multiple network_resources_table_sections. Transmission of the NRT is accomplished by partitioning the table into a series of sections. The last_section_number field in the MPEG-2 Private Section header specifies the number of sections used.

As an example, the Resource Descriptor could be used to provide the internet address of a server an interactive service, or a remote Program element in another MPEG-2 Transport Stream.

Note: In the formula specified in the ATSC Data Broadcast Standard [1], the constant 27MHz should equal 27,000,000 for calculations.

The following data structure describes the NRT encapsulated in a networkResourcesTable_section (Table 8.3).

Table 8.3 Network Resources Table

Field Name	No. of Bits	Field Value	Notes
networkResourceTable_section() {			
table_id	8	0xD1	
section_syntax_indicator	1	1b	
private_indicator	1	1b	
reserved	2	11b	
private_section_length	12		The length in bytes starting from table_id_extension field to the end of the networkResourceTable_section.
table_id_extension	16	0xFFFF	ATSC reserved
reserved	2	11b	
version_number	5		
current_next_indicator	1	1b	
section_number	8		
last_section_number	8		
network_resource_table_bytes() {			Network Resource Table.
resource_descriptor_count_in_section	8		Number of resource descriptors listed in this NRT section.
if(resource_descriptor_count_in_section > 0){			
for (j=0; j<resource_descriptor_count_in_section; j++){			
compatibility_descriptor() {			
compatibilityDescriptorLength	16		Length is from descriptorCount to the end of the compatibility_descriptor.
descriptorCount	16		
for (i=0; i<descriptorCount; i++){			
descriptorType	8		
descriptorLength	8		
specifierType	8		
specifierData	24		
model	16		
version	16		
subDescriptorCount	8		
for(j=0; j<subDescriptorCount;j++) {			
subDescriptor() {			
subDescriptorType	8		
subDescriptorLength	8		
for(k=0; k<subDescriptorLength; k++){			
additionalInfo	8		User private data.
}			
}			
}			

}			
}			
dsmccResourceDescriptor() {			
commonDescriptorHeader() {			
resourceRequestId	16		The only value supported is 0xFFFF. This value indicates that this field is not applicable. The original purpose of this field is to link the assignment of a resource to an initial request.
resourceDescriptorType	16		This field identifies the resource descriptor.
resourceNum[15,14]	2	11b	Network assigned resource number.
resourceNum[13…0]	14		
associationTag	16		Matches the association_tag value in one or more Tap structures of the DST.
resourceFlags	8	0x03	
resourceStatus	8	0x04	
resourceLength	16		
resourceDataFieldCount	16		
if (resourceDescriptorType == 0xFFFF) {			
typeOwnerId	24		
typeOwnerValue	24		
}			
}			
resourceDescriptorDataFields() {			
descriptors go here. See [4] for the format of this structure.			The descriptor may be a deferredMPEGProgramElement, an IPResourceDescriptor, an IPV6ResourceDescriptor or a URLResourceDescriptor.
}			
}			
}			End of resource descriptor loop
}			end of if resource_descriptor_count_in_section > 0
privateDataLength	16		
for (i=0; i<privateDataLength; i++) {			
privateDataByte	8		
}			
}			
CRC_32	32		
}			

8.5 Service Discovery and Binding—Putting It All Together

Figure 8.2 illustrates the use of the Service Description Framework. Two applications are discovering and acquiring the resources each application requires for proper operation. The PSIP

Data Service announcement is assumed to have occurred and is outside of the scope of the diagram.

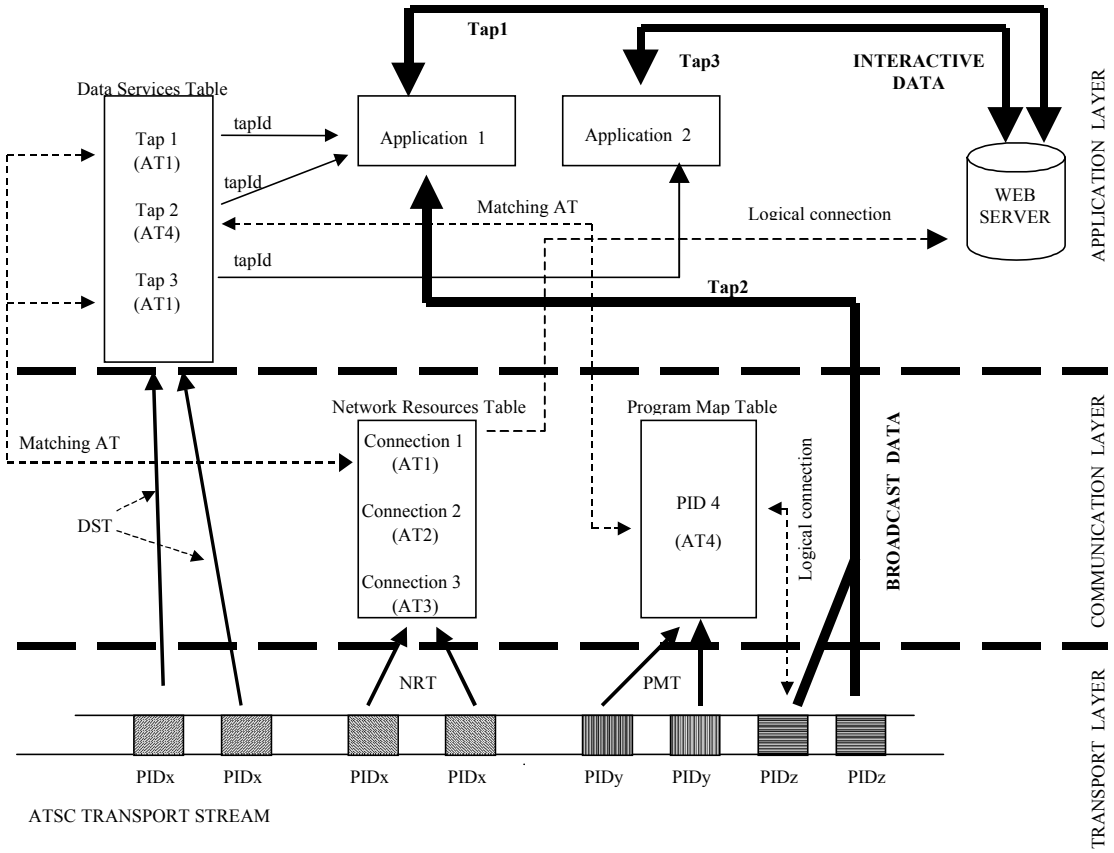


Figure 8.2 Service description block diagram..

In Figure 8.2, two applications, Application 1 and Application 2, are both simultaneously active in the data receiver. Each application is responsible for acquiring its required resources. Application 1 requires two resources and Application 2 requires a single resource. The resources for each application are described in the Data Services Table (DST) carried on PIDx. The resources are identified by the Tap structures contained in the DST where each Tap structure has a unique tapId. The PID on which the DST is carried, in this case PIDx, is identified in the Program Map Table (PMT) where the elementary_PID is set to PIDx and the stream_type 0x95. Additionally, the DST PID is also located in the PSIP Service Location Descriptor.

For Application 1, an external device supplies the first resource, identified by Tap 1. The second resource used by Application 1, identified by Tap 2, is contained within the current MPEG-2 Transport Stream. For Application 2, its only resource is identified by Tap 3. The resource is the same external logical connection as used by Tap 1; however, it is a different resource. By using distinct Taps, references to different data components transmitted on the same logical connection can be accomplished. This concept is further illustrated as follows.

In Figure 8.2, Tap 1 and Tap 3 identify different data resources located on the same logical connection. For example, Tap 1 specifies the id of a data resource, say id1 and Tap 3 specifies the id of another data resource, say id2. Both data resources reside on the same external logical

connection identified by the URL "http://www.domainName.com". Data from id1 is subsequently passed to Application1 and data from id2 is passed to Application 2.

The DST also indicates that Tap 1 and Tap 3 reside on an external logical connection via the resourceLocation field, and thus, their resource location must be discovered via the Network Resource Table (NRT). The NRT is carried in the same PID, PIDx, as the DST and is uniquely identified by a separate table_id. The same logical connection is located in the NRT by using the associationTag found in the Tap structure. In Figure 8.2, Tap 1 and Tap 3 share the same associationTag value AT1 and thus, the same logical connection, Connection 1. Taps using the same associationTag field can either be associated with the same application or with different applications. In Figure 8.2, Connection 1 provides the information necessary to determine that the logical connection is to the Web Server and thus interactive data is available via this mechanism.

Application 1's second resource is located in the current MPEG-2 Transport Stream. The second resource used by Application 1 is identified by Tap 2 and the resource's binding association tag is AT4. Tap 2 indicates that the resource can be found in the Program Map Table. By locating AT4 in the PMT, the logical connection, in this case PIDz, is identified and PIDz's information is subsequently passed to Application 1. Note that the use of the association_tag provides the additional benefit of preserving the stream/application association when re-multiplexing (change in the elementary PID) occurs.

9. BUFFER MODEL

Buffer models are used for the delivery of asynchronous, synchronous and synchronized Data Services. A buffer model provides a framework for controlled and timely delivery of data from an emission station to receivers. Hence, both emission and reception sides of an ATSC communication channel must take the buffer model into account either to insert data into the ATSC Transport multiplex or to extract data from the multiplex.

The purpose of a buffer model at the emission side is to provide multiplexers with the tools for scheduling insertion of data bytes into an ATSC Transport multiplex in accordance with the Data Service requirements (allocated bandwidth). On one hand, the data server in the emission station must take into account the fact that receivers have finite amount of memory. On the other hand, the buffer model represents the minimum amount of memory that must be dedicated to the acquisition of data bytes in a data receiver. Therefore, for a given transmission rate, buffers represent a window of opportunity in time for inserting data into the ATSC Transport multiplex.

The reception end buffer model helps data receivers extract and reconstruct data from the ATSC Transport multiplex before being forwarded to higher application levels. The buffer model also specifies the minimum requirements of data throughput in data receivers.

There are three buffer models, one for asynchronous, one for synchronous and one for synchronized Program elements of a Data Service. The buffer models for asynchronous, synchronous and synchronized data elementary streams include a Transport Buffer and a Smoothing Buffer as defined in Section 2.4.2.3 of ISO/IEC 13818-1 [2] with the additional feature that a leak rate is specified at the output of the Smoothing Buffer. The buffer model for synchronized data elementary streams features an additional buffer called the Data Elementary Buffer. The presence of this buffer allows reconstruction of "Data Access Units", that is data aggregated from multiple PES packets or sections and making up an individual unit to be presented at a given instant of the System Time Clock. The concept of Data Access Unit is not present in the buffer models for asynchronous elementary streams as these buffer models do not include a Data Elementary Buffer. Synchronous elementary streams have the concept of 16 bit

synchronous data access units that are leaked from the smoothing buffer at a piecewise continuous rate signaled by either the increment field or the ES_rate field (as appropriate).

The buffer model associated with a Program element conveying the Service Description Framework data will be the same as the one used for asynchronous data elementary streams (Program elements).

9.1 Buffer Model for Asynchronous Data Elementary Streams

The buffer model described in this section is the Transport System Target Data Decoder buffer model for asynchronous Program elements.

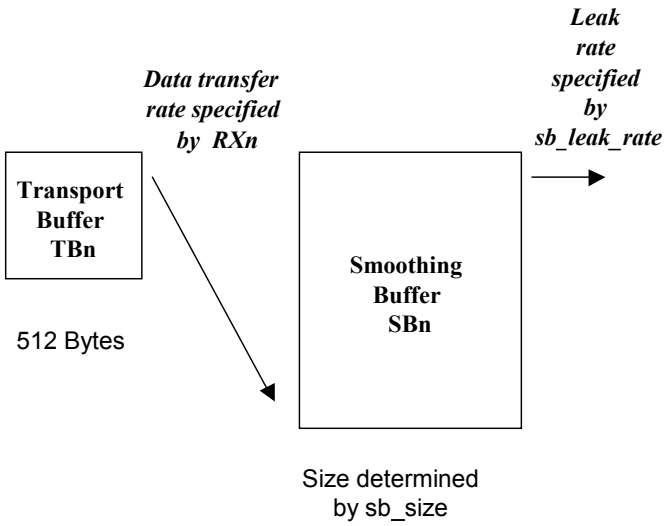


Figure 9.1 Transport system target data decoder model for asynchronous data elementary streams.

This buffer model is applicable to data elementary streams (Program elements) of stream_type value 0x0B, 0x0D, and 0x95 (SDF protocol encapsulation values 0x01, 0x03, 0x04, 0x0C and 0x0D). The buffer model comprises a Transport Buffer of 512 bytes followed by a Smoothing buffer of 4500 or 10000 bytes, depending on the Data Service profile. (The Data Service profile is specified in the data_service_descriptor.) On the data receiver side, both buffers are not allowed to overflow. The purpose of the Transport Buffer is to prevent bursty transmission of Transport Stream packets belonging to the same data elementary streams, thereby providing a more uniform chance for Transport Stream packets of other Program elements to be transmitted as well. The purpose of the Smoothing Buffer is to compensate for variations of time elapsed between transmission of consecutive Transport Stream packets belonging to the same Program element. These time variations are due to the fact that other Program elements have been multiplexed into the same Transport Stream. The output data rate defined by the Data Service profile is re-constructed at the output of the Smoothing Buffer.

Complete Transport Stream packets containing data from data elementary stream *n*, as indicated by its PID, are passed to the Transport Buffer for stream *n*, TB*n*. This includes duplicate Transport Stream packets and packets with no payloads. Transfer of any byte from the System Target Data Decoder input to TB*n* is considered instantaneous.

The data transfer rate from the Transport Buffer to the Smoothing Buffer is equal to 1.2 times the rate specified by the Data Service profile. Therefore, the instantaneous rate at which the

Smoothing Buffer fills is larger than the output leakage rate. All bytes that enter the Transport Buffer also exit it.

Only the Transport Stream packet payload bytes belonging to an MPEG-2 Private Section, a DSM-CC section, or a DSM-CC addressable section are transferred into the Smoothing Buffer. Other bytes are not and may be used to control the system. Duplicate Transport Stream packets are not delivered to the Smoothing Buffer. Any data transfer from the Transport Buffer to the Smoothing Buffer is considered instantaneous.

When there is data present in the Smoothing Buffer, bytes are removed from this buffer at the specified leak rate. All bytes that enter the Smoothing Buffer also exit it.

9.2 Smoothing Buffer Descriptor

The size of the Smoothing Buffer, SBS*n*, and the output bitrate of the Smoothing Buffer may be specified by means of the smoothing_buffer_descriptor. The location of this descriptor depends on the value of the PSIP service_type field associated with the data service, as described below:

- In the Program element (Elementary Stream or inner) information descriptor loop of the PMT for Data Services of type 0x02, 0x03 and 0x04. In this case, the value of the sb_leak_rate and sb_size apply to the data elementary stream (Program element) referenced by the Transport Stream PID value elementary_PID. Only Transport Stream packets of this Program element enter the Smoothing Buffer.
- In the extended program information descriptor loop (First or outer loop) of the PMT for Data Services of type 0x04. In this case, the smoothing_buffer_descriptor value sb_leak_rate and sb_size apply to all Program elements of the program referenced in the PMT by the same program_number value. All Transport Stream packets of the program enter the Smoothing Buffer.
- In an EIT-k table and in association with a particular event for Data Services of type 0x02 or 0x03. In this case, the smoothing_buffer_descriptor value of sb_leak_rate and sb_size apply to all Program elements of the Data Service. All Transport Stream packets of the Data Service enter the Smoothing Buffer.
- In the DET-k table and in association with a particular event for Data Services of type 0x02, 0x03 or 0x04. In this case, the value of sb_leak_rate and sb_size apply to all Program elements of the Data Service. All Transport Stream packets of the Data Service enter the Smoothing Buffer.

In the default case when no Smoothing Buffer Descriptor is present, the leak rate and size of the Smoothing Buffers are specified by the profiles contained in the data_service_descriptor, namely:

- 384 kbps and 4500 bytes for profile G1
- 3.84 Mbps and 4500 bytes for profile G2
- 19.2 Mbps and 10000 bytes for profiles G3 and A1

These values apply to all data elementary streams (Program elements) belonging to the Data Service. All Transport Stream packets belonging to the Data Service enter the Smoothing Buffer.

Table 9.1 Smoothing Buffer Descriptor Location Matrix

service_type	PMT Program Element Level	PMT Program Level	EIT	DET
0x02	*		*	*
0x03	*		*	*
0x04	*	*		*

9.3 Maximum Bitrate Descriptor

The maximum bit rate may be specified by means of the maximum_bitrate_descriptor. The value of the bit rate in this descriptor indicates an upper bound of the bit rate, including Transport overhead that will be encountered in a data elementary stream, a program or a Data Service. This descriptor may be located in the following structures:

- In the ES information descriptor loop of the PMT for Data Services of type 0x02, 0x03 and 0x04. In this case, the value of maximum_bitrate applies to the data elementary stream (Program element) referenced by the Transport PID value elementary_PID.
- In the extended program information descriptor loop of the PMT for Data Services of type 0x04. In this case, the value maximum_bitrate applies to all Program elements of the program referenced in the PMT by the same program_number value.
- In an EIT-k table and in association with a particular event for Data Services of type 0x02 or 0x03. In this case, the value of maximum_bitrate applies to all Program elements of the Data Service.
- In the DET-k table and in association with a particular event for Data Services of type 0x02, 0x03 or 0x04. In this case, the value of maximum_bitrate applies to all Program elements of the Data Service.

Table 9.2 Maximum Bitrate Descriptor Location Matrix

service_type	PMT Program Element Level	PMT Program Level	EIT	DET
0x02	*		*	*
0x03	*		*	*
0x04	*	*		*

In the default case where the maximum bit rate descriptor is not used, the maximum bit rate is the maximum Data Service bit rate of 19.392656 Mbits/sec.

9.4 Buffer Model for Synchronous Data Elementary Streams

The Transport System Target Data Decoder model for synchronous data elementary streams (Program elements) is similar to the T-STD described in the previous section for asynchronous data elementary streams. Both the Transport Buffer TBn and the Smoothing Buffer SBn are not allowed to overflow.

This buffer model is applicable to data elementary streams (Program elements) of stream_type value 0xC2 (SDF protocol encapsulation values 0x06, 0x08 and 0x0A).

As opposed to the asynchronous case, the MPEG-2 Transport packets convey only PES packets. Therefore, only Transport Stream packet payload bytes belonging to a PES packet are transferred into the Smoothing Buffer. Other bytes are not and may be used to control the

system. Duplicate Transport Stream packets are not delivered to the Smoothing Buffer. Any data transfer from the Transport Buffer to the Smoothing Buffer is considered instantaneous.

If the PES packet that has been in buffer SBn the longest includes a PTS field in its header bytes, the first 16-bit data access unit in this PES packet is removed from buffer SBn at the instant specified by the value of the PTS field. Subsequent 16-bit access data units are removed at the rate specified either by the increment field in the synchronous_data_header structure or the ES_rate field in the PES packet header. The increment field is used for leak rates below 9.0 Mbps and the ES_rate field is used for leak rates above 9.0 Mbps. Bytes that are not part of a 16-bit data_access_unit are removed instantaneously, discarded and may be used to control the system. Data Access Units in any subsequent PES packets that do not have any PTS fields are delivered according to the leak rate specified by the increment field or the ES_rate field of their respective PES packet. The leak rate specified by the increment field or the ES_rate field shall never exceed the leak rate specified by the data service profile or by the sb_leak field in a smoothing descriptor, if present for this data service or synchronous data elementary stream.

For any ATSC virtual channel featuring a synchronous data elementary stream, there must be a Program Element that includes Program Clock References (PCR) time stamps. This is such that PTS in the PES packet header can be referenced to instants of the timeline at which the 16-bit data access units must be delivered.

9.5 Buffer Model For Synchronized Data Services

9.5.1 T-STD for Synchronized Program Elements

The Transport System Target Data Decoder model for synchronized data elementary streams (Program elements) includes the components of the T-STD described in the previous section for synchronous data elementary streams.

This buffer model is applicable to data elementary streams (Program elements) of stream_type value 0x06 and 0x14 (SDF protocol_encapsulation values 0x02, 0x05, 0x07 and 0x09). Streaming synchronized data elementary streams are conveyed in PES packets. Non-streaming synchronized data elementary streams are conveyed in the synchronized Download protocol fragmented in DSM-CC sections. The latter case corresponds to the isolated transmission of data to be synchronized with a video or audio elementary streams.

The MPEG-2 Transport Stream packets convey either PES packets or DSM-CC sections conveying the synchronized Download protocol. Only Transport Stream packet payload bytes belonging to a PES packet or to the DSM-CC section of a synchronized download are transferred into the Smoothing Buffer. Other bytes are not and may be used to control the system. Duplicate Transport Stream packets are not delivered to the Smoothing Buffer. Any data transfer from the Transport Buffer to the Smoothing Buffer is considered instantaneous.

In a synchronized Data Service, the reference timing is provided in another elementary stream (a video, audio or another data elementary stream) belonging to the same program. The field PCR_PID in the PSIP Service Location Descriptor (SLD) and the PMT identifies the Transport PID of the Program element conveying this time reference. Any synchronized data elementary stream includes Presentation Time Stamps (PTS). As opposed to the T-STD for synchronous data elementary streams, the buffer model includes a Data Elementary Stream buffer DEBn which allows reconstruction and synchronization of data access units with the reference video/audio/data Program element. (See Figure 9.2.)

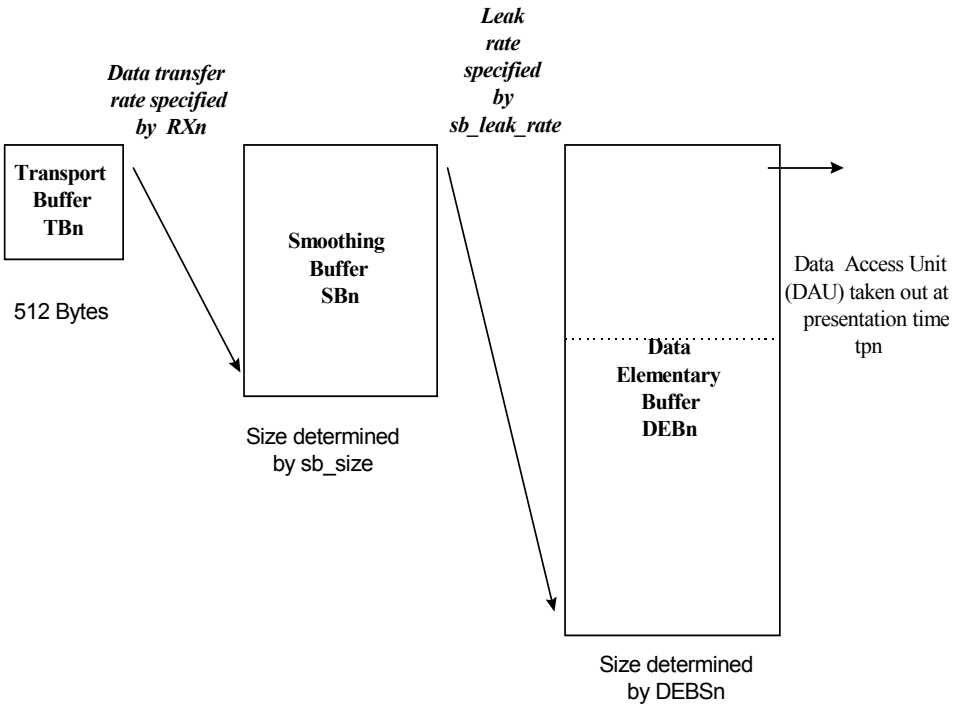


Figure 9.2 System target data decoder buffer model for synchronized data services.

Input data to the Data Elementary Stream buffer are the bytes originating from the Smoothing Buffer SBn. Any synchronized data elementary stream conveys Data Access Units. A Data Access Unit (DAU) will be equal to the in order concatenation of

- One or more PES packets starting with the first PES packet bearing a PTS field up to and not including the next PES packet bearing a PTS field
- One or more DSMCC_section sections of the same version, starting with the section with section_number value 0 bearing a PTS field, conveying a single DSM-CC data module (same table_id_extension value)

A DAU includes the PES header bytes or the DSMCC_section header bytes, respectively. Each DAU includes a distinct Presentation Time Stamp. The purpose of the Presentation Time Stamp is to associate the complete DAU with a particular instant of the concurrent audio-visual event. How the DAU is taken out of the DEBn buffer depends on whether the data elementary stream is either a streaming synchronized or a non-streaming synchronized data elementary stream.

The Transport System Target Decoder model defined in the ATSC Data Broadcast Standard [1] is a hypothetical buffer model assuming instantaneous decoding of the Data Access Units.

All bytes at the output of the Smoothing Buffer SBn, associated with a synchronized data elementary stream denoted by index *n*, are passed to the Data Elementary Buffer, referred to as DEBn. For synchronized protocol encapsulations based on PES packetization, bytes from the PES packet header and bytes from the synchronized_data_packet structure are removed instantaneously and discarded and may be used to control the system. For synchronized protocol encapsulations based on DSM-CC section packetization, only data bytes of the payload of DSMCC_section structures with table_id value 0x3C enter buffer DEBn. The header bytes from such DSM-CC section are removed instantaneously and discarded and may be used to control the system. Bytes from the DSM-CC section CRC_32 field or checksum field are also removed

instantaneously and discarded and may be used to verify the integrity of the data. Furthermore, bytes from the dsmccDownloadDataHeader message header and the first 6 bytes of the DownloadDataBlock message (representing a moduleId, a moduleVersion, a reserved and a blockNumber field, respectively) are discarded and may be used to control the system. If there is PES packet or DSM-CC section data in SBn and buffer DEBn is not full, the data is transferred from SBn to DEBn at a rate defined by sb_leak. If the buffer DEBn is full, no data is removed from the Smoothing Buffer SBn. When there is no PES packet or DSM-CC section bits in SB*n*, no data is removed from SBn. The buffer DEBn will not be allowed to overflow.

For the Data Elementary Buffer DEBn, all data for the Data Access Unit that has been in the buffer longest are removed instantaneously at time tpn assuming instantaneous decoding. The Presentation Time tpn is specified by the PTS field of the DAU. The Transport System Target Decoder model defined in this specification is a hypothetical buffer model assuming instantaneous decoding of the Data Access Units.

Data Access Units of a streaming synchronized data elementary stream may be taken as early as the Presentation Time Stamp of the previous transmitted Data Access Unit. This use of the Presentation Time Stamps is consistent with usage of time stamps for I and P frames in video.

Data Access Units of a non-streaming synchronized data elementary stream may be taken out of the DEBn buffer once it has been received in full in the Data Elementary Stream buffer. However, additional auxiliary information may be conveyed to delay the time at which the Data Access Unit is to be taken out of the buffer. Such auxiliary information may be a common delay value for all Data Access Units in the non-streaming data elementary stream. In this case, the delay value indicates how much time before its associated PTS a DAU must be taken out of the buffer. Consequently, the delay value also represents the largest time required to decode a Data Access Unit of the data elementary stream.

The buffer DEBn must not be allowed to overflow. Overflow is characterized by a loss of information due to a shortage of unoccupied space in the elementary buffer.

In the case of a streaming synchronized data elementary stream or in the case of a non-streaming synchronized data elementary stream, the buffer DEBn must not underflow when a Data Access Unit is removed from the buffer at its associated decoding time. Underflow is characterized by a data access unit not having been fully received in the elementary buffer by the time it must be taken out of the buffer model. Satisfying a non-underflow condition translates to quality of service since such condition requires that the data must be transmitted and received in a timely fashion.

Note that in the case of a streaming synchronized data elementary stream consisting of Data Access Units conveyed each across multiple PES packets, the underflow condition requires a constraint on how the data elementary stream must be constructed. In this scenario, the difficulty comes from the fact that the size of the last Data Access Unit of the data elementary stream cannot be determined as there is no other DAU following (no other PES packet with a PTS in its header). Therefore, it is necessary to signal the end of the last DAU in one way or another. The simplest method is to require that the last PES packet of the streaming synchronized data elementary stream be an empty PES packet (no payload) with a PTS in its header.

The minimal value for DEBSn must be equal to 120120 bytes. This buffer size corresponds to three times the maximum amount of data that can be transmitted in 16.683333 milliseconds ($1/60^{th}$ of a second) at a 19.2 Mbits/sec data transfer rate (highest possible data rate). The assumption is made that a nominal DAU size corresponds to the maximum number of bytes that can be transmitted in $1/60^{th}$ of a second. The size of a nominal DAU is therefore equal to 19.2 Mbits/sec $\times$ 1001 / ($8 \times 60 \times 1000$) = 40040 bytes and the nominal Data Access Units frequency

is 59.94 Hz. The data conveyed in a nominal DAU can therefore be viewed as being in association with a particular DTV video field. The purpose of defining a nominal Data Access Unit is to provide a reference point to the specification of Data Elementary Stream Buffer size. The size of the Data Elementary Stream Buffer is such that it can hold up to three nominal Data Access Units. Such a choice for DEBSn allows for one nominal DAU to be flushed out of the buffer while a second re-assembled nominal DAU is held ready and a third nominal DAU is being acquired.

Data Access Units must be separated in time by no less than 5.561111 millisecond (corresponding to a Data Access Unit frequency of 3 times the nominal Data Access Unit frequency of 59.94 Hz used above). Hence, the minimum leak rate required at the output of the Data Elementary Stream buffer is equal to 172.8 Mbits/sec for a level 1 Data Service (see next section for definition of levels). The value 172.8 Mbits/sec is obtained by dividing the full capacity of the data elementary stream buffer (120120 × 8 bits) by 5.561111 milliseconds. This case corresponds to the extreme case where the size of a DAU is the size of the DEBn buffer and the previous and next DAU are 5.561111 milliseconds later.

Figure 9.2 illustrates the buffer arrangement for synchronized Data Services. As mentioned earlier, the size of the Transport Buffer TBn is equal to 512 bytes and the size and leak rate of the Smoothing Buffer SBn is specified by the Data Service profile or the sb_size and sb_leak_rate field values, respectively. The Data Access Units are input to the Data Elementary Buffer DEBn. In accordance with the T-STD model, the Data Access Units are taken out of the Data Elementary Buffer instantaneously and at presentation time tpn specified in the PTS field of the DAU.

9.5.2 Data Service Levels

A Data Service level is specific to a particular value of DEBSn, the size of the data elementary stream buffer. The level also determines the throughput required for transferring synchronized data in the receiver. A level 1 Data Service must correspond to a DEBSn size equal to 120120 bytes as described in the previous section. The Level 4 , Level 16 and Level 64 Data Services are defined in term of the level 1 DEBSn value as follows:

- Level 1 (multiplicative factor is equal to 1) will signal a DEBSn value equal to120120 bytes
- Level 4 (multiplicative factor is equal to 4) will signal a DEBSn value equal to 480480 bytes.
- Level 16 (multiplicative factor is equal to 16) will signal a DEBSn value equal to 1921920 bytes.
- Level 64 (multiplicative factor is equal to 64) will signal a DEBSn value equal to 7687680 bytes.

The nominal Data Access Unit size for level 1, level 4, level 16 and level 64 will be 40040 bytes, 160160 bytes, 640640 bytes and 2562560 bytes, respectively.

Data service profiles and levels are directly related to the acquisition and throughput capability of a data receiver. The value of the Data Service level is signaled along with the Data Service profile in the data_service_descriptor.

When there are more than one synchronized data elementary stream in a Data Service, the size DEBSn of the DEBn buffer is split uniformly among all the synchronized data elementary streams, regardless whether there may be non-streaming or streaming synchronized Program elements. In cases when such uniform splitting results in fractional byte allocation, the buffer

size assigned to each synchronized data elementary stream will be rounded down to the nearest integer byte size. The purpose of this splitting is to allow multiple applications of the same Data Service to be run concurrently in the receiver. Without such splitting process, the amount of DEBn memory that is required in a data receiver would be unbounded. Hence, prior to acquiring synchronized data, the receiver must evaluate the number of synchronized streams published in the Data Service so the Data Elementary Stream can be split accordingly. The following pseudo-C code is an example of how the number L of synchronized streams in a Data Service can be calculated. The program calculates L by keeping a record of all distinct association tag values for synchronized stream. Note that the synchronized stream are not necessarily in the current MPEG-2 program as the association tag may be pointing to an external Virtual Channel in the same ATSC Transport Stream through the Network Resource Table (NRT). This situation can occur only if both the Virtual Channel of the data service and the external service reference the same PCR_PID in their program definition.

9.5.2.1 Synchronized Stream Calculation Pseudo Code

```
int int sync'ed_AT_list[65536];
srv.sync'edPrgmElement_DEBsize = 0;
L = 0;
for( m = 0; m < applicationCount; m++)
{
    for(n = 0; n < tapsCount; n++)
    {
        if(   (srv.app[m].tap[n].protocol_encapsulation != 02)
        &&  (srv.app[m].tap[n].prococol_encapsulation != 05)
        &&  (srv.app[m].tap[n].protocol_encapsulation != 07)
        &&  (srv.app[m].tap[n].protocol_encapsulation != 09))
        {
            // skip  non-synchronized Program elements
            continue;
        }
        foundAT = FALSE;
        currentAT = srv.app[m].tap[n].AT;
        currentAT |= (srv.app[m].tap[n].resource_location << 16) | srv.app[m].tap[n].AT;
        for( j = 0; j < L; j++)
        {
            if( currentAT == sync'ed_AT_list[ j ] )
            {
                foundAT = TRUE;
                break;
            }
        }
        if( foundAT == FALSE)
        {
            sync'ed_AT_list[ L++ ] = currentAT;
        }
    }
}
if( L > 0 )
{
    srv.sync'edPrgmElement_DEBsize = floor(  DEBSn[srv.level ] / L );
}
```

Annex A:
Descriptor Location Matrix

1. SERVICE DESCRIPTION FRAMEWORK (SDF) DESCRIPTOR LOCATIONS

Table A1 identifies the possible locations for each descriptor defined or used by the ATSC Data Broadcast Standard [1] Service Description Framework (SDF).

Table A1 SDF Descriptor Location Matrix

Descriptor	PMT	SLD	EIT	DET	LTST	DST	NRT
association_tag_descriptor	*(1)						
data_service_descriptor			*	*	*		
pid_count_descriptor			*	*	*		
download_descriptor						*	
multiprotocol_encapsulation_descriptor						*	
compatibility_descriptor						*(2)	*(2)
dsmccResourceDescriptor							*
deferredMPEGProgramElement							*(3)
IPResourceDescriptor							*(3)
IPV6ResourceDescriptor							*(3)
URLResourceDescriptor							*(3)
smoothing_buffer_descriptor	*(4)		*	*	*		
maximum_bitrate_descriptor	*(4)		*	*	*		

Notes:

1) The assoctiation_tag_descriptor can only appear in the Program element (elementary stream) descriptor loop of the Program Map Table (PMT).
2) The compatibility_descriptor is not constrained.
3) These descriptors are all components and located within the dsmccResourceDescriptor.
4) This descriptor can appear in either descriptor loop of the PMT.

1.1 Data Download Protocol Descriptor Location Matrix

Table A2 identifies the possible locations for the descriptors used by the data download protocol defined in ATSC Data Broadcast Standard [1].

Table A2 Data Download Protocol Descriptor Location Matrix

Descriptor	DII	DSI	DC	DDB
module_link_descriptor	*			
CRC32_descriptor	*			
group_link_descriptor		*		
compatibility_descriptor	*(1)	*(1)		

Notes:

1) The compatibility_descriptor has been constrained to a size of 0x0000 bytes.

Annex B:
ATSC Data Broadcast Encapsulation Protocols

1. SCOPE

This annex describes the messages that are used for the data broadcast encapsulation protocols defined in the ATSC Data Broadcast Standard. The syntax used to describe the messages follows a "pseudo-code" approach, with the intent of making it easy to understand what fields the messages contain and what they mean. Subfields of a field are indicated by indentation level, rather than relying on curly brackets to delimit them. In many places a table in the standard indicates a field with a variable number of bytes by giving a "count" subfield, followed by a "for loop" subfield for the individual bytes of the field. The pseudo-code descriptions in this annex typically describe such fields by just naming them and indicating that they have a variable number of bytes. To see the details of how such a field should be coded, see the standard itself or the more detailed syntax descriptions that appear in the main body of these Implementation Guidelines.

1.1 Data Download Protocol Messages

1.1.1 DSM-CC Section

The messages of the data download protocol are all encapsulated in *DSM-CC Sections* (defined in Section 9.2.2 of the DSM-CC standard [4]), which are a special case of MPEG-2 *private sections* (defined in Section 2.4.4.10 of the MPEG-2 Systems standard [2]). Table B1 describes the format of the DSM-CC section, as used in the data download protocol.

Table B1 DSM-CC Section Format

Field	No. of Bits	Value
DSMCC_section(){		
DSMCC_section_header(){		
table_id	8	0x3B for DII, DSI, or DC message; 0x3C for DDB message
section_syntax_indicator	1	'0' if checksum used for error checking; '1' if CRC_32 used for error checking
complement_indicator	1	'1' if section_syntax_indicator is '0'; '0' if section_syntax_indicator is '1'
reserved	2	'11'
dsmcc_section_length	12	number of bytes in section after this field; must not exceed 4093
table_id_extension	16	bits 15-0 of transaction_id (identification and updated_flag) for DSI, DII, or DC messages; moduleId for DDB messages
reserved	2	'11'
version_number	5	'00000' for DSI, DII, or DC message; least significant 5 bits of moduleVersion for DDB messages
current_next_indicator	1	'1' (always current)
section_number	8	0x00 for DSI, DII, or DC message; least significant 8 bits of blockNumber for DDB messages
last_section_number	8	0x00 for DSI, DII, or DC message. Maximum of section numbers that appear in DDB messages for same version of same module. (Note that this may not be the section number of the last block of the module, since section numbers wrap around after they reach 255.)
}		
<body of section>	variable	DSM-CC message
checksum or **CRC_32**	32	checksum or CRC_32, as indicated by section_syntax_indicator
}		

1.1.1.1 DSM-CC Message Header and DSM-CC Download Data Header

The download control messages of the data download protocol (DSI, DII, or DC) all consist of the *DSM-CC Message Header* (defined in Section 2 of the DSM-CC standard [4]), followed by the message body. Table B2 describes the format of the DSM-CC Message Header, as used in the data download protocol.

Table B2 DSM-CC Message Header

Field	No. of Bits	Value
dsmccMessageHeader(){		
protocolDiscriminator	8	0x11
dsmccType	8	0x03
messageId	16	0x1006 for DSI message; 0x1002 for DII message; 0x1005 for DC message
transactionId	32	transaction_id (See Section 6.1.2.)
reserved	8	0xFF
adaptationLength	8	length of adaptationHeader, in bytes
messageLength	16	total length of the message after this field, including dsmccAdaptationHeader if any
if (adaptationLength > 0)		
dsmccAdaptationHeader()	variable	
}		
}		

The ATSC Data Broadcast Standard does not specify any use for the Adaptation Header in the control messages, so the adaptationLength will typically be 0.

The DDB messages of the data download protocol consist of the *DSM-CC Download Data Header* (defined in Section 7.2.2.1 of the DSM-CC standard [4]), followed by the message body. Table B3 describes the format of the DSM-CC Download Data Header, as used in the data download protocol.

Table B3 DSM-CC Download Data Header

Field	No. of Bits	Value
dsmccDownloadDataHeader()		
protocolDiscriminator	8	0x11
dsmccType	8	0x03
messageId	16	0x1003 for DDB message
downloadId	32	download_id (See Section 6.1.2.)
reserved	8	0xFF
adaptationLength	8	length of adaptationHeader, in bytes
messageLength	16	total length of the message after this field, including dsmccAdaptationHeader if any
if (adaptationLength > 0)		
dsmccAdaptationHeader()	variable	
}		
}		

The primary difference between the DSM-CC Download Data Header and the DSM-CC Message Header is that the Download Data Header has the downloadId in the position where the

DSM-CC Message Header has the transactionId. (See Section 6.1.2 for descriptions of these two fields.)

For the asynchronous download scenarios the adaptationLength will typically be 0. However, for the synchronized download scenario the dsmccAdaptationHeader contains the presentation time stamp (PTS) values that are used for synchronization. In that case the dsmccAdaptationHeader has the format described in Table B4, and has length 8 bytes (64 bits).

Table B4 DSM-CC Adaptation Header

Field	No. of Bits	Value
dsmccAdaptationHeader()		
adaptationType	8	0x04 for adaptation header with PTS value
if (adaptationType = 0x04){		
reserved	16	0xFFFF
'0010'	4	'0010'
PTS[32 … 30]	3	leftmost 3 bits of 33-bit PTS value
marker_bit	1	'1'
PTS[29 ... 15]	15	next 15 bits of 33-bit PTS value
marker_bit	1	'1'
PTS[14 … 0]	15	rightmost 15 bits of 33-bit PTS value
marker_bit	1	'1'
}		
else{		
adaptationDataBytes	variable	
}		
}		

1.1.1.2 Download Server Initiate Message

Table B5 describes the Download Server Initiate (DSI) message format.

Table B5 Download Server Initiate Message

Field	No. of Bits	Value
DownloadServerInitiate(){		
dsmcc_section_header()	64	
dsmccMessageHeader()	96 + A	
serverId	160	0x FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
compatibility_descriptor	16	0x0000
privateDataLength	16	length in bytes of the rest of the message after this field (but not including the CRC/checksum)
GroupInfoIndication()		
numberOfGroups	16	number of groups in the supergroup
for (i=0; i<numberOfGroups; i++)		
groupId	32	transactionId of DII message for group
groupSize	32	cumulative size in bytes of all modules in the group
groupCompatibility()	variable	compatibility descriptor
groupInfoLength	16	total length in bytes of groupInfoBytes
groupInfoBytes	variable	list of descriptors
}		
}		
CRC_32 or **checksum**	32	CRC_32 or checksum, as indicated by section_syntax_indicator in dsmcc_section_header()
}		

The “A” term in the size of the dsmccMessageHeader() field is the size of the adaptation header in the dsmccMessageHeader(). As noted earlier, it will typically be 0.

The compatibility descriptor structure in the groupCompatibility field can be used to specify features that the receiver must possess in order to utilize the data in the group. It is described in detail in Section 6.1 of the ATSC Data Broadcast Standard. Often the groupCompatibility field will simply have the value 0x0000, indicating that no compatibility conditions are specified.

The only descriptor defined in the ATSC Data Broadcast Standard for the groupInfo descriptor list is the group_link_descriptor, which allows multiple groups to be linked together into a linked list. It is described in Table B6.

Table B6 Group Link Descriptor

Field	No. of Bits	Value
group_link_descriptor(){		
descriptor_tag	8	0xB8
descriptor_length	8	5
position	8	0x00 indicates group is first group in list 0x01 indicates group is intermediate group in list 0x02 indicates group is last group in list
group_id	32	transactionId of DII message for next group in list (ignored when position = 0x02)
}		

1.1.1.3 Download Info Indication Message

Table B7 below describes the Download Info Indication (DII) message format.

Table B7 Download Info Indication Message

Field	No. of Bits	Value
DownloadInfoIndication(){		
dsmcc_section_header()	64	
dsmccMessageHeader()	96 + A	
downloadId	32	downloadId for download scenario
blockSize	16	length in bytes of blocks in DDB messages
windowSize	8	0x00 (not used)
ackPeriod	8	0x00 (not used)
tCDownloadWindow	32	0x00000000 (not used)
tCDownloadScenario	32	timeout period in microseconds for entire download scenario
compatibilityDescriptor()	2	0x0000 (not used)
numberOfModules	2	total number of modules in the group
for (i=0; i<numberOfModules; i++){		
moduleId	16	ModuleId
moduleSize	32	total size of module in bytes; set to 0 for unbounded module
moduleVersion	8	module version number
moduleInfoLength	8	length in bytes of moduleInfoBytes
moduleInfoBytes	variable	list of descriptors
}		
privateDataLength	16	length in bytes of privateDataBytes
privateDataBytes	variable	private data
CRC_32 or **checksum**	32	CRC_32 or checksum, as indicated by section_syntax_indicator in dsmcc_section_header()
}		

The "A" term in the size of the dsmccMessageHeader() field is the size of the adaptation header in the dsmccMessageHeader(). As noted earlier, it will typically be 0.

Note that since the block size is specified outside the module loop in the DII message, the block size must be the same for all modules in a group, except that the last block of each module may be smaller.

Typically the privateDataLength will be 0x0000, and there will be no privateDataBytes.

The ATSC Data Broadcast Standard specifies just two descriptors for the moduleInfo descriptor list in the DII messages, the module_link_descriptor and the CRC32_descriptor, described in Tables B8 and B9.

The module_link_descriptor allows multiple modules to be linked together in a linked list. This can be used to define a logical module which is much larger than the module size limit.

Table B8 Module Link Descriptor

Field	No. of Bits	Value
module_link_descriptor(){		
descriptor_tag	8	0xB4
descriptor_length	8	3
position	8	0x00 indicates module is first module in list 0x01 indicates module is intermediate module in list 0x02 indicates module is last module in list
module_id	16	moduleId of next module in list (ignored when position = 0x02)
}		

The CRC32 descriptor can be used to provide the calculation of a CRC over a complete data module, instead of just over individual blocks.

Table B9 CRC32 Descriptor

Field	No. of Bits	Value
CRC32_descriptor(){		
descriptor_tag	8	0xB5
descriptor_length	8	4
CRC_32	32	32-bit CRC calculated over entire module
}		

1.1.1.4 Download Cancel Message

Table B10 below describes the Download Control (DC) message format.

Table B10 Download Cancel Message

Field	No. of Bits	Value
downloadCancel(){		
dsmcc_section_header()	64	
dsmccMessageHeader()	96 + A	
downloadId	32	downloadId for download scenario
moduleId	16	moduleId of last processed DDB message at time of cancellation
blockNumber	16	blockNumber of last processed DDB message at time of cancellation
downloadCancelReason	8	reason code for cancelling
reserved	8	0xFF
privateDataLength	16	length in bytes of privateDataBytes
privateDataBytes	variable	private data
CRC_32 or **checksum**	32	CRC_32 or checksum, as indicated by section_syntax_indicator in dsmcc_section_header()
}		

The "A" term in the size of the dsmccMessageHeader() field is the size of the adaptation header in the dsmccMessageHeader(). As noted earlier, it will typically be 0.

The DSM-CC standard defines about 20 reason codes, but about the only one that really makes sense in a broadcast environment is 0x04 (rsnFatal), announcing that a fatal error of some sort has occurred on the server side.

1.1.1.5 Download Data Block Message

Table B11 describes the Download Data Block (DDB) message format.

Table B11 Download Data Block Message

Field	No. of Bits	Value
DownloadDataBlock(){		
dsmcc_section_header()	64	
dsmccDownloadDataHeader()	96 + A	
moduleId	16	identifier of module to which this block belongs
moduleVersion	8	current version of module to which this block belongs
reserved	8	0xFF
blockNumber	16	position of this block within module; block 0 is first block of module
blockDataBytes	variable	data in block
}		

The "A" term in the size of the dsmccDownloadDataHeader() field is the size of the adaptation header in the dsmccMessageHeader(). As noted earlier, it will typically be 0 for the asynchronous download scenarios and 8 for a synchronized download scenario.

1.1.1.6 Encapsulating Download Messages into MPEG-2 Transport Stream Packets

All of the messages used in the data download protocol are encapsulated in DSM-CC sections. Thus, to describe how these messages get into an MPEG-2 transport stream, all that is needed is to describe how DSM-CC sections are packed into MPEG-2 Transport Stream packets. The method for doing this is the same as the method for packing MPEG-2 PSI sections or MPEG-2 private sections into Transport Stream packets, as specified in the MPEG-2 Systems standard.

Table B12 describes the structure of an MPEG-2 Transport Stream packet, in the context of the encapsulation of DSM-CC sections.

Table B12 MPEG-2 Transport Stream Packet Header

Field Name	No. of Bits	Field Value
sync_byte	8	0x47
transport_error_indicator	1	'1' indicates packet error detected during transmission '0' indicates no known error
payload_unit_start_indicator (PUSI)	1	'1' indicates packet contains start of section '0' indicates that it does not
transport_priority	1	'1' means packet has higher priority '0' means packet has lower priority
PID	13	identifies data elementary stream to which packet belongs
transport_scrambling_control	2	'00' means not scrambled; other values are user defined
adaptation_field_control	2	'00' is reserved '01' means no adaptation field, payload only '10' means only adaptation field, no payload '11' means adaptation field and payload
continuity_counter	4	0x0 to 0xF, incremented for each successive packet with a given PID, with wrap-around from 0xF to 0
adaptation_field (if adaptation field control is '00' or '10')	variable	may contain PCR values and other useful stuff
pointer_field (if payload unit start indicator is '1')	8	0 to 182, indicating byte position in packet containing first byte of first section that starts in packet, starting with 0 for the first byte after the pointer_field.
payload	N * 8	

Assume we have a sequence of DSM-CC sections that are part of the same download scenario, and they are to be put into a sequence of Transport Stream packets having the PID designated for that download scenario. The first section can start anywhere within the payload of the first Transport Stream packet. It then proceeds to fill up the rest of the payload of that Transport Stream packet and as many additional successive Transport Stream packets as are needed to hold it all. If it does not end exactly on a Transport Stream packet boundary, then the next section can start immediately after it the point where it ended, or the rest of that Transport Stream packet can be stuffed with 0xFF bytes, and the next section can start in the next Transport Stream packet in the sequence.

Each Transport Stream packet containing the start of a section will have the payload_unit_start_indicator set, and the first byte of what would otherwise be the payload becomes a pointer_field, telling where in the Transport Stream packet to find the start of the first section that starts in the packet. (If sections are very short, a Transport Stream packet may contain the start of two or more sections.) See Figure B1.

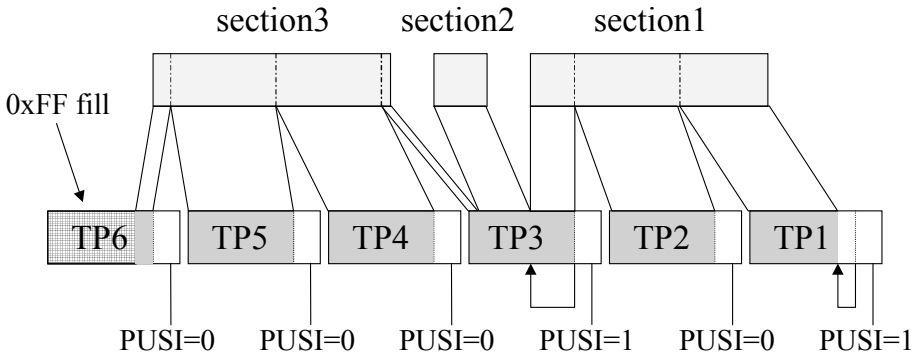


Figure B1 Packing DSM-CC Sections into MPEG-2 Transport Stream Packets

All messages of a single download scenario must go in a single data elementary stream, and nothing else may go in that data elementary stream; i.e., all the MPEG-2 Transport Stream packets carrying the messages must have the same value for the PID (packet id) in the packet header. Moreover, no other Transport Stream packets in the transport stream may have that PID value.

Annex C:
Sample Encapsulations

1. DATA DOWNLOAD PROTOCOL EXAMPLE

The example given in this annex shows the encoding of a simple two-layer asynchronous, non-streaming Data Service via the non-flow controlled download scenario. For this example, a Data Service Program element is envisioned that carries textual information in two different languages. Two groups are used, one carrying a text payload in English, the other carrying the equivalent text payload in French to illustrate grouping. The data payload is a simple text message: "The quick brown fox jumped over the lazy dog." (or "The rapide renard brun saute au dessus du chien qui se repose."[6]) and in each case, the text is carried via a single DownloadDataBlock message (that fits within a single MPEG2 Transport Stream packet). Each DII only carries information about the one module within the group. Data constructs are represented first by tables similar to those above (with the appropriate values filled in), followed by the hex representation of the MPEG-2 Transport Stream packet.

In this example, the transmission order will be: DSI, DII (English), DDB (English), DII (French), DDB (French). All the MPEG-2 Transport Stream packet will be located on PID 0x00FF.

1.1 Sample Tables

1.1.1 Sample DSI

See Table C1 for the sample DSI.

Table C1 Sample DSI

Field Name	No, of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	Indicates no error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x00FF	Arbitrarily chosen for use in this example.
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0x0	1st packet
pointer_field	8	0x00	Section starts immediately after this field.
table_id	8	0x3B	0x3B = DSM-CC section with DownloadServerInitiate messages.

6 The authors of this Implementation Guide acknowledge our imperfect translation to the French language.

section_syntax_indicator	1	0b	0b indicates that the checksum/CRC32 field contains a checksum.
private_indicator	1	1b	This field is set to the complement of the section_syntax_indicator flag.
reserved	2	11b	Reserved fields are always filled in with '1's.
dsmcc_section_length	12	0x049	The number of bytes from table_id_extension through the last byte of the CRC32/Checksum field. The maximum value is 4093.
table_id_extension	16	0x0000	This field is set to the 2 least significant bytes of the transactionId.
reserved	2	11b	
version_number	5	00000b	This field is set to 00000b.
current_next_indicator	1	1b	This field is set to 1b, indicating that the information in the DSI is always current.
section_number	8	0x00	This field is set to 0x00
last_section_number	8	0x00	This field is set to 0x00
protocolDiscriminator	8	0x11	DSMCC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1006	DownloadServerInitiate message.
transactionId(){			Resulting value = 0x40000000.
originator subfield	2	10	The server sets this field.
version subfield	14	00000000000000b	Initial value.
identification subfield	15	000000000000000b	All zeros
updated subfield	1	0b	Initial value.
}			End of sub-fields in transactionId
reserved	8	0xFF	
adaptationLength	8	0x00	No adaptation header present.
messageLength	16	0x0034	Length of message immediately following this field up to, but not including the checksum/CRC32 field.
serverId	20Bytes	0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	This field is set to 20 bytes, each containing the value 0xFF.
compatibilityDescriptorLength	16	0x0000	Always 0x0000.
privateDataLength	16	0x001C	Length in bytes of the following GroupInfoIndication structure.
numberOfGroups	16	0x0002	2 Groups.

groupId	32	0x80000002	This field is equal to the transactionId of the DII message that describes the group (English text in this case).
groupSize	32	0x0000002D	This field indicates the cumulative size in bytes of all the modules in the group. (Sum of all the moduleSize's in the DII describing this group.)
groupCompatibilityDescriptorLength	16	0x0000	No groupCompatibilityDescriptor
groupInfoLength	16	0x0000	No descriptors in loop
groupId	32	0x80000004	This field is equal to the transactionId of the DII message that describes the group (French text in this case).
groupSize	32	0x0000003D	This field indicates the cumulative size in bytes of all the modules in the group. (Sum of all the moduleSize's in the DII describing this group.)
groupCompatibilityDescriptorLength	16	0x0000	No groupCompatibilityDescriptor
groupInfoLength	16	0x0000	No descriptors in the loop.
privateDataLength	16	0x0000	No private data.
checksum/CRC32	32	0x00000000	Checksum set to zero signals that no checksum was calculated.

The following is the hex representation of the Table C1 DSI message in an MPEG-2 Transport Stream packet:

```
4740 FF10 003B 7049  0000 C100 0011 0310
0680 0000 00FF 0000  34FF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FF00 0000
1C00 0280 0000 0200  0000 2D00 0000 0080
0000 0400 0000 3D00  0000 0000 0000 0000
00FF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF
```

1.1.2 Sample DII for Group 1 (English)

Table C2 lists the sample DII for Group 1 (English).

Table C2 Sample DII for Group 1 (English)

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	Indicates no error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x00FF	Arbitrarily chosen for use in this example.
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0x1	2nd packet
pointer_field	8	0x00	Section starts immediately after this field.
table_id	8	0x3B	0x3B = DII.
section_syntax_indicator	1	0b	Checksum used.
private_indicator	1	1b	Complement of the section_syntax_indicator flag.
reserved	2	11b	Reserved fields are always filled in with ‘1’s.
dsmcc_section_length	12	0x033	The number of bytes from table_id_extension through the last byte of the CRC32/Checksum field.
table_id_extension	16	0x0002	The 2 least significant bytes of the transactionId.
reserved	2	11b	
version_number	5	00000b	This field is set to 00000b.
current_next_indicator	1	1b	DII current.
section_number	8	0x00	Always 0x00.
last_section_number	8	0x00	Always 0x00.
protocolDiscrimiator	8	0x11	DSM-CC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1002	DII.
transactionId	32	0x80000002	Originator subfield = 10b Version subfield = 0x0000 Identifier subfield = 000000000000001b Updated subfield = ‘0’
reserved	8	0xFF	
adaptationLength	8	0x00	Set to 0x00.
messageLength	16	0x001E	Length of message immediately following this field up to, but not including the checksum/CRC32 field.
downloadId	32	0x00000000	carousel_id.
blockSize	16	0x0FE2	4066 bytes (non-synchronized case)
windowSize	8	0x00	Not used in broadcast.
ackPeriod	8	0x00	Not used in broadcast.

tCDownloadWindow	32	0x00000000	Not used in broadcast.
tCDownloadScenario	32	0x00000000	Timeout period in microseconds for entire download.
compatibilityDescriptorLength	16	0x0000	Always 0x0000
numberOfModules	16	0x0001	Number of modules in the loop to follow.
moduleId	16	0x0002	Unique identifier for the module.
moduleSize	32	0x0000002D	Size of the module payload (don't count DDB syntax or checksum) in bytes when bounded.
moduleVersion	8	0x00	Version number of the module.
moduleInfoLength	8	0x00	Length of the moduleInfo loop to follow.
privateDataLength	16	0x0000	User private.
checksum	32	0x00000000	Checksum set to zero signals that no checksum was calculated.

The following is the hex representation of the Table C2 DII message in an MPEG2 Transport Stream packet:

```
4740 FF11 003B 7039  0002 C100 0011 0310
0280 0000 02FF 0000  1E00 0000 000F E200
0000 0000 0000 0000  0000 0000 0100 0200
0000 2D00 0000 0000  0000 00FF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF
```

1.1.3 Sample DDB (for English module)

Table C3 lists the sample DDB (for English module).

Table C3 Sample DDB (for English module)

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	No error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x00FF	Arbitrarily chosen for use in this example.
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0x2	3rd packet.
pointer_field	8	0x00	Section starts immediately after this field.

table_id	8	0x3C	0x3C = DSM-CC section with Download data messages.
section_syntax_indicator	1	0b	checksum.
private_indicator	1	1b	Complement of the section_syntax_indicator flag.
reserved	2	11b	Reserved fields are always filled in with '1's.
dsmcc_section_length	12	0x048	Number of bytes from the start of table_id_extension through the last byte of the CRC32/Checksum field.
table_id_extension	16	0x0002	This field will convey a copy of the moduleId value found in the DDB.
reserved	2	11b	
version_number	5	0x00	This field will convey a copy of the least significant 5-bits of the moduleVersion value found in the DDB.
current_next_indicator	1	1b	Data is always current.
section_number	8	0x00	LSB of blockNumber.
last_section_number	8	0x00	This field is set to the maximum value that is encoded in the section_number field.
protocolDiscrimiator	8	0x11	DSM-CC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1003	DDB
downloadId	32	0x00000000	Identifier of the download scenario in progress. In the case of a data carousel scenario of the data download protocol, this field is set to the value of the carousel_id if it is specified in the download_descriptor in the associated Data Service Table.
reserved	8	0xFF	
adaptationLength	8	0x00	Set to 0x00 when no PTS field is present.
messageLength	16	0x0033	Length of message immediately following this field up to, but not including the checksum/CRC32 field.
moduleId	16	0x0002	The moduleId field identifies to which module this block belongs.
moduleVersion	8	0x00	Initial version.
reserved	8	0xFF	
blockNumber	16	0x0000	Position of block with the module. Zero corresponds to the first block.
blockDataByte	N*8	The quick brown fox jumped over the lazy dog.	Actual data from file/module.
checksum	32	0x00000000	Checksum set to zero signals that no checksum was calculated.

The following is the hex representation of an MPEG2 Transport Stream packet carrying the Table C3 DDB:

```
4740 FF12 003C 7048  0002 C100 0011 0310
0300 0000 00FF 0000  3300 0200 FF00 0054
6865 2071 7569 636B  2062 726F 776E 2066
6F78 206A 756D 7065  6420 6F76 6572 2074
6865 206C 617A 7920  646F 672E 0000 0000
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF
```

1.1.4 Sample DII for Group 1 (French)

Table C4 lists the sample DII for Group 1 (French).

Table C4 Sample DII for Group 1 (French)

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	Indicates no error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x00FF	Arbitrarily chosen for use in this example.
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0x3	4th packet
pointer_field	8	0x00	Section starts immediately after this field.
table_id	8	0x3B	0x3B = DII.
section_syntax_indicator	1	0b	Checksum used.
private_indicator	1	1b	Complement of the section_syntax_indicator flag.
reserved	2	11b	
dsmcc_section_length	12	0x033	The number of bytes from table_id_extension through the last byte of the CRC32/Checksum field.
table_id_extension	16	0x0004	The 2 least significant bytes of the transactionId.
reserved	2	11b	Reserved fields are always filled in with ‘1’s.
version_number	5	00000b	This field is set to 00000b.
current_next_indicator	1	1b	DII current.
section_number	8	0x00	Always 0x00.

last_section_number	8	0x00	Always 0x00.
protocolDiscriminator	8	0x11	DSM-CC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1002	DII.
transactionId	32	0x80000004	Originator subfield = 10b Version subfield = 0x0000 Identifier subfield = 000000000000010b Updated subfield = '0'
reserved	8	0xFF	
adaptationLength	8	0x00	Set to 0x00
messageLength	16	0x001E	Length of message immediately following this field up to, but not including the checksum/CRC32 field.
downloadId	32	0x00000000	carousel_id.
blockSize	16	0x0FE2	4066 bytes (non-synchronized)
windowSize	8	0x00	Not used in broadcast.
ackPeriod	8	0x00	Not used in broadcast.
tCDownloadWindow	32	0x00000000	Not used in broadcast.
tCDownloadScenario	32	0x00000000	Timeout period in microseconds for entire download.
compatibilityDescriptorLength	16	0x0000	Always 0x0000
numberOfModules	16	0x0001	Number of modules in the loop to follow.
moduleId	16	0x0003	Unique identifier for the module.
moduleSize	32	0x0000003D	Size of the module payload (don't count DDB syntax or checksum) in bytes when bounded.
moduleVersion	8	0x00	Version number of the module.
moduleInfoLength	8	0x00	Length of the loop to follow.
privateDataLength	16	0x0000	User private.
checksum	32	0x00000000	Checksum set to zero signals that no checksum was calculated.

The following is the hex representation of the Table C4 DII message in an MPEG2 Transport Stream packet:

```
4740 FF13 003B 7033  0004 C100 0011 0310
0280 0000 04FF 0000  1E00 0000 000F E200
0000 0000 0000 0000  0000 0000 0100 0300
0000 3D00 0000 0000  0000 00FF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF
```

1.1.5 Sample DDB (for French module)

Table C5 gives the sample DDB (for French module).

Table C5 Sample DDB (for French module)

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	No error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x00FF	Arbitrarily chosen for use in this example.
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0x4	5th packet.
pointer	8	0x00	Section starts immediately after this field.
table_id	8	0x3C	0x3C = DSM-CC section with Download data messages.
section_syntax_indicator	1	0b	checksum.
private_indicator	1	1b	Complement of the section_syntax_indicator flag.
reserved	2	11b	Reserved fields are always filled in with ‘1’s.
dsmcc_section_length	12	0x058	Number of bytes from the start of table_id_extension through the last byte of the CRC32/Checksum field.
table_id_extension	16	0x0003	This field will convey a copy of the moduleId value found in the DDB.
reserved	2	11b	
version_number	5	0x00	This field will convey a copy of the least significant 5-bits of the moduleVersion value found in the DDB.
current_next_indicator	1	1b	Data is always current.
section_number	8	0x00	LSB of blockNumber.
last_section_number	8	0x00	This field is set to the maximum value that is encoded in the section_number field.
protocolDiscrimiator	8	0x11	DSM-CC message.
dsmccType	8	0x03	U-N Download message.
messageId	16	0x1003	DDB

downloadId	32	0x00000000	Identifier of the download scenario in progress. In the case of a data carousel scenario of the data download protocol, this field is set to the value of the carousel_id if it is specified in the download_descriptor in the associated Data Service Table.
reserved	8	0xFF	
adaptationLength	8	0x00	Set to 0x00 when no PTS field is present.
messageLength	16	0x0043	Length of message following this field including the dsmccAdaptationHeader as specified by the adaptationLength.
moduleId	16	0x0003	The moduleId field identifies to which module this block belongs.
moduleVersion	8	0x00	
reserved	8	0xFF	
blockNumber	16	0x0000	Position of block with the module. Zero corresponds to the first block.
blockDataByte	N*8	The rapide renard brun saute au dessus du chien qui se repose.	Actual data from file/module.
checksum	32	0x00000000	Checksum set to zero signals that no checksum was calculated.

The following is the hex representation of an MPEG2 Transport Stream packet carrying the Table C5 DDB:

```
4740 FF14 003C 7058  0003 C100 0011 0310
0300 0000 00FF 0000  4300 0300 FF00 0054
6865 2072 6170 6964  6520 7265 6E61 7264
2062 7275 6E20 7361  7574 6520 6175 2064
6573 7375 7320 6475  2063 6869 656E 2071
7569 7365 2072 6570  6F73 652E 0000 0000
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF  FFFF FFFF
```

1.2 DSM-CC Addressable Section Example

The example given next shows the encoding of a UDP/IP datagram where the UDP/IP datagram's data payload is the text message, "The quick brown fox jumped over the lazy dog." The sentence is carried via a single DSMCC_addressable_section that fits within a single MPEG2 Transport Stream packet. The UDP datagram is sent to UDP destination port 4800 (0x12C0) from source port 1387 (0x056B). The IP destination address is 224.7.8.9, and the source IP address is 192.168.1.220. The deviceId is the RFC 1112 converted MAC address for the

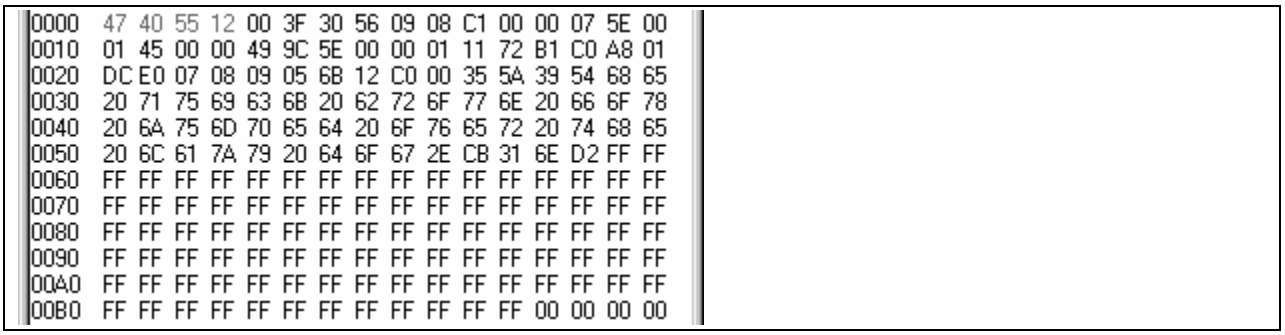
multicast IP destination address. The section protection is provided by a CRC32 calculation, and signaled accordingly in the DSMCC_addressable_section by the protection_indidcator set to 0b.

Table C6 DSM-CC Addressable Section Example

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	No error.
payload_unit_start_indicator	1	1b	Section starts in this packet.
transport_priority	1	0b	No special priority.
PID	13	0x0055	
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	01b	01 = No adaptation field, payload only.
continuity_counter	4	0010b	3rd packet.
pointer	8	0x00	Section starts immediately after this field.
DSMCC_addressable_section() {			
table_id	8	0x3F	
'0'	1	0b	
error_detection_type	1	0b	0b indicates that the checksum/CRC32 field contains a CRC32.
reserved	2	11b	
section_length	12	0x056	The number of bytes from deviceID_6 to end of the DSMCC_addressable_section.
deviceID [7..0]	8	0x09	Least significant byte of the deviceID (i.e., byte 6).
deviceID [15..8]	8	0x08	Byte 5 of the deviceID
reserved	2	11b	
payload_scrambling_control	2	00b	00b = Unscrambled.
address_scrambling_control	2	00b	00b = Unscrambled.
LLC_SNAP_flag	1	0b	0b indicates that the datagram is IP and not LLC encapsulated.
'1'	1	1b	
section_number	8	0x00	Only 1 section per DSMCC_addressable_section for IP.
last_section_number	8	0x00	Only 1 section per DSMCC_addressable_section for IP.
deviceID[23..16]	8	0x07	Byte 4 of the deviceID.
deviceID [31..24]	8	0x5E	Byte 3 of the deviceID.
deviceID [39..32]	8	0x00	Byte 2 of the deviceID.
deviceID [47..40]	8	0x01	Byte 1 of the deviceID (most significant byte).

if (LLC_SNAP_flag == '1') {			
LLC_SNAP()	8 * 8		Included if the LLCSNAP_flag is set to 1b.
} else {			
datagram_data	8		IP Header, 20 bytes, starts with 0x45 0x00...0x07, 0x08, 0x09. UDP Header, 8 bytes, 0x05 6B...0x5A, 0x39. Text = The quick brown fox jumped over the lazy dog.
}			
if (section_number == last_section_number) {			
for (j=0;j<N2;j++) {			
stuffing_byte	8		No stuffing bytes.
}			
}			
checksum/CRC_32	32	0xCB316ED2	CRC32
}			

The following is the hex representation of an MPEG-2 Transport Stream packet carrying a UDP/IP datagram encapsulated in DSM-CC addressable section.



1.3 Data Piping Example

Table C7 illustrates the coding of a sentence using the Data Piping protocol and the stuffing of the packet using the MPEG-2 adaptation field. The entire sentence is contained with a single MPEG Transport Stream packet.

Table C7 Data Piping Example

Field Name	No. of Bits	Field Value	Notes
sync_byte	8	0x47	Sync byte.
transport_error_indicator	1	0b	No error.
payload_unit_start_indicator	1	0b	
transport_priority	1	0b	No special priority.
PID	13	0x0055	
transport_scrambling_control	2	00b	00b = Not scrambled.
adaptation_field_control	2	11b	11 = Adaptation field followed by payload.
continuity_counter	4	0000b	1st packet.
adaptation_field_length	8	0x2D	The number of bytes (45) immediately following the adapation_field_length to the end of the adaptation field. 0 = Insertion of a single stuffing byte. The example requires a total of 46 bytes of stuffing. One byte is occupied by the adaptation length field and the other byte is filled by the adaptation flags field. The remaining 44 bytes are located in the stuffing_bytes loop.
if (adaptation_field_length > 0) {			
flags	8	0x00	Set to zero.
for (i=0; i<N; i++) {			N is 44 bytes.
stuffing_byte	N * 8		44 bytes of 0xFF.
}			
} // end if adaptation_field_length			
payload	N * 8	See the next column.	138 bytes of data equal to “IG authors: John R. Mick Jr. – SkyStream Networks, Rich Chernock - IBM, Regis Crinon – Intel, Edwin Heredia - Samsung, Art Allison – NAB\n” where \n=0x0D 0x0A.
}			

The following is the hex representation of an MPEG-2 Transport Stream packet containing the Data Piping encapsulated sentence.

```
47 00 55 30 2d 00 ff ff ff ff ff ff ff ff ff ff G.U0-.....
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
ff ff 49 47 20 61 75 74 68 6f 72 73 3a 20 4a 6f ..IG authors: Jo
68 6e 20 52 2e 20 4d 69 63 6b 20 4a 72 2e 20 2d hn R. Mick Jr. -
20 53 6b 79 53 74 72 65 61 6d 20 4e 65 74 77 6f  SkyStream Netwo
72 6b 73 2c 20 52 69 63 68 20 43 68 65 72 6e 6f rks, Rich Cherno
63 6b 20 2d 20 49 42 4d 2c 20 52 65 67 69 73 20 ck - IBM, Regis
43 72 69 6e 6f 6e 20 2d 20 49 6e 74 65 6c 2c 20 Crinon - Intel,
45 64 77 69 6e 20 48 65 72 65 64 69 61 20 2d 20 Edwin Heredia -
53 61 6d 73 75 6e 67 2c 20 41 72 74 20 41 6c 6c Samsung, Art All
69 73 6f 6e 20 2d 20 4e 41 42 0d 0a             ison - NAB..
```

Annex D:
Service Description Framework Example

1. SCOPE

This annex illustrates the PSIP and SDF infrastructure that is associated with the examples of Annex C. See Figure D1.

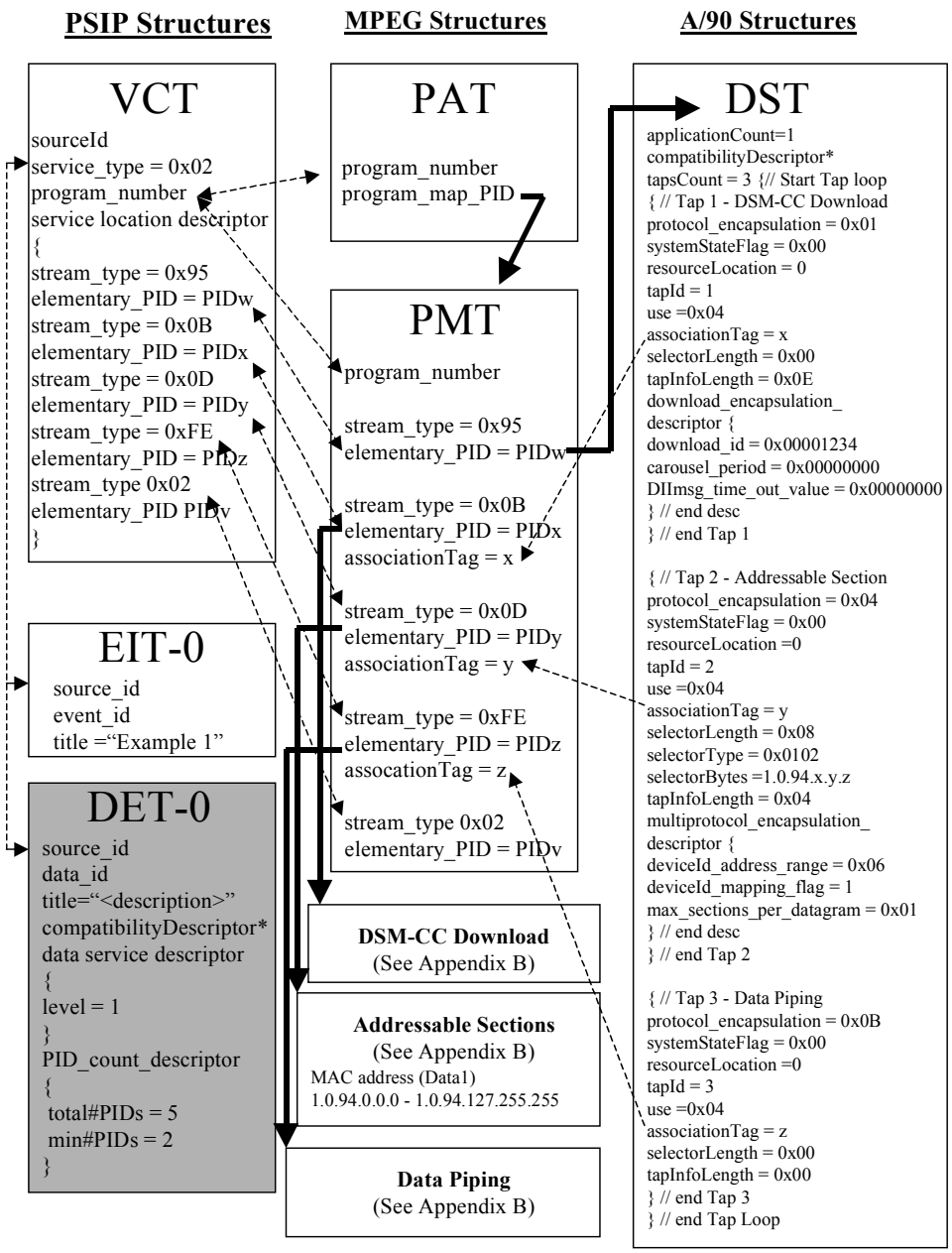


Figure D1 Example ATSC table hierarchy.

1.1 ATSC Table Hierarchy Discussion

Figure D illustrates one Virtual Channel conveying a DSM-CC data download protocol stream, a single IP Multicast stream, and a Data Piping stream each as separate Program elements that are part of a single Data Service. All three Program elements comprising the Data Service are being related to a video elementary stream. Hence in this example, the PSIP service_type is 0x02. Also, it is assumed that the Program elements are conveyed in the same MPEG-2 program as the Program element conveying the Service Description Framework (SDF) information. This example does not have any requirement for Internet network access or for external Program elements. Therefore, Figure D does not include a Network Resource Table (NRT).

In Figure D, the dashed lines show field correspondences. Fields linked by a dashed line have the same value. Solid, thick lines refer to elementary_PID based identification of the MPEG-2 Transport Stream packets conveying a particular Program element—like the solid line from the program_map_PID field in the Program Association Table (PAT) to the Program Map Table (PMT).

The Virtual Channel Table (VCT) is transmitted in MPEG-2 Transport Stream packets referenced by the PID value 0x1FFB. The source_id field associated with the Virtual Channel in the VCT allows for the identification of the instances of the Event Information Tables (EIT-k's) announcing the audio-visual event schedule. It also identifies the instances of the Data Event Tables (DET-k's) announcing the schedule of the associated Data Service. The location of the EIT-k's and DET-k's in the MPEG-2Transport Stream is specified in the PSIP Master Guide Table (MGT). The instances of the DET-k's conveying the Data Service schedule include a title text structure that may be reported in the on screen display of the Program Guide.

The schedule of the audio-visual and associated data event appears ultimately in an instance of an EIT-0 or DET-0 table, respectively. The audio, video and data Program elements belong to the same Virtual Channel so the audio-visual event can be enhanced by the Data Service.

The PSIP Service Location Descriptor (SLD) in the VCT is used to aggregate the Program elements making up the Virtual Channel. The Virtual Channel must include one and only one Program Element of stream_type 0x95. The elementary_PID value PIDw associated with the Program element of stream_type value 0x95 specifies the MPEG-2 Transport Stream packets conveying the Data Service Table and the Network Resources Table (not included) in the Virtual Channel.

The elementary_PID value PIDx of stream_type value 0x0B identifies the MPEG-2 Transport Stream packets conveying the DSM-CC data download protocol encapsulation. The elementary_PID value PIDy of stream_type value 0x0D identifies the MPEG-2 Transport Stream packets conveying the DSM-CC addressable sections where the IP Multicast data datagrams are encapsulated. The elementary_PID value PIDz of stream_type value 0xFE (user specified) identifies the Transport Stream packets conveying the Data Piping protocol.

The elementary_PID values PIDw, PIDx, PIDy, and PIDz are also listed in the PMT. The Virtual Channel may include additional elementary_PID values for video and audio streams. This concept is shown in Figure D where a video elementary stream of type 0x02 is signaled in the SLD and the PMT. Since in this example the Data Service is related to a video event, the service_type value in the VCT is equal to 0x02.

The Program Association Table (PAT) in the ATSC Transport mutliplex identifies the PID value of the MPEG-2 Transport Stream packets conveying the Program Map Table (PMT). The PMT includes an MPEG-2 program featuring the video, audio and data elementary streams aggregated by the PSIP Virtual Channel. The MPEG-2 program_number in the PAT and the PMT match the program_number associated with the Virtual Channel listed in the VCT.

The DST consists of one application featuring three Tap structures. The Tap structures include an associationTag allowing identification of the location (the elementary PID value) of

the Program element within the ATSC Transport multiplex. The elementary_PID value associated with the Program element is obtained by matching the associationTag value in the Tap structure with the associationTag value in the Association Tag Descriptor within the PMT.

The selector bytes of the Tap include the MAC Multicast address of the datagrams conveying the SDP protocol. The Tap information loop includes a Multiprotocol Encapsulation Descriptor to signal the number of active bytes used in the deviceId fields of the DSM-CC Addressable Sections. The deviceId_address_range field value is set to 0x06 to indicate that all deviceId bytes are active. The deviceId_mapping_flag is set to 1 to indicate that deviceId fields represent a MAC address.