

ATSC Standard: Transport Stream File System Standard

Note that this documents is past the customary 5-year review point. No update of the document is in process.

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

The Advanced Television Systems Committee, Inc., is an international, non-profit organization developing voluntary standards for digital television. The ATSC member organizations represent the broadcast, broadcast equipment, motion picture, consumer electronics, computer, cable, satellite, and semiconductor industries.

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 Council 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 170 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	6
1.1 Organization	6
2. REFERENCES	6
2.1 Normative References	6
2.2 Informative References	7
3. DEFINITIONS AND STRUCTURES	7
3.1 Compliance Notation	7
3.2 Acronyms and Abbreviations	8
3.3 Global Terms	8
3.4 Section and Data Structure Syntax Notation	11
3.5 Elements with Undefined Semantics	12
3.6 Code Points (Informative)	12
4. INTRODUCTION (INFORMATIVE)	13
5. STRUCTURES	16
5.1 Carousel NSAP Address	16
5.2 Interoperable Object Reference (IOR) Format	17
5.3 References within the Same TSFS	18
5.4 References to Remote Objects	22
5.5 BIOP Message Formats	25
5.6 Uniform Resource Identifiers	33
6. OBJECTINFO DESCRIPTORS	33
6.1 Descriptor Identification and location	34
6.2 Content Type Descriptor	34
6.3 Time Stamp Descriptor	35
7. TRANSPORT	35
7.1 DSM-CC Message Header	36
7.2 Transport of Service Gateway IOR	37
7.3 Transport of Module Delivery Parameters	39
7.4 Semantics of TransactionId	42
7.5 Signaling of Transport Stream File Systems	43
7.6 Private Usage Collision Avoidance	45
Annex A: Carousel Design (Informative)	47

Annex B: Object and File System Acquisition (Informative)	48
1. INTRODUCTION.....	48
2. LOCAL OBJECT ACQUISITION.....	48
3. REMOTE OBJECT RESOLUTION.....	51
4. TRANSPORT STREAM FILE SYSTEM ACQUISITION.....	53
5. URI RESOLUTION.....	54
Annex C: TSFS Objectives (Informative)	51
1. OBJECTIVES FROM RFP.....	57
2. ACHIEVEMENT OF OBJECTIVES.....	57
2.1 Name Spaces	57
2.2 Selective Acquisition and Browsing	58
2.3 Carriage within Virtual Channels	58
2.4 Optimization Opportunities	58
2.5 Meta-Data Extensibility	59

Index of Tables and Figures

Table 5.1 Carousel NSAP Address Syntax	16
Table 5.2 Definition of the IOR Structure	18
Table 5.3 Value of typeId_bytes	18
Table 5.4 BIOPProfileBody Structure	20
Table 5.5 MessageSelector Structure	22
Table 5.6 Definition of the LiteOptionsProfileBody Structure	23
Table 5.7 Value of kind_data Bytes	24
Table 5.8 Syntax of Directory Message Format	26
Table 5.9 Object Kind and Binding Type Codes	29
Table 5.10 Syntax of File Message Format	31
Table 6.1 objectInfo descriptors and locations	34
Table 6.2 Syntax of ContentTypeDescriptor	34
Table 6.3 Syntax of TimeStamp_descriptor	35
Table 7.1 Syntax of a DSI Message	37
Table 7.2 Syntax of BIOP: ServiceGatewayInfo	38
Table 7.3 Syntax of a DII Message	39
Table 7.4 TransactionId Sub-fields	42
Table 7.5 DST Tap Referencing a Transport Stream File System	44
Table 7.6 TSFS Selector for Tap Referencing a TSFS	45
Table C.7 TSFS Objectives	57
Figure 4.1 Graphical encapsulation overview and relation to other standards.	13
Figure 4.2 (a) Directory object (b) File object.	14
Figure 7.1 Encapsulation and fragmentation of BIOP messages.	36
Figure B1 Resolving an object from its IOR with BIOP profile body.	49
Figure B2 Use of Tap for locating a program element.	50
Figure B3 Finding service gateway from NSAP address (no remultiplexing case).	52
Figure B4 Finding service gateway from NSAP address (remultiplexed case).	53
Figure B5 How to acquire objects in a simple directory structure.	54
Figure B6 URIs derived from TSFS directory structure.	55

Transport Stream File System Standard

1. SCOPE

This standard was prepared by the Advanced Television Systems Committee (ATSC) Technology Group on Distribution (T3). The document was approved by T3 on 15 August 2002 for submission by letter ballot to the full ATSC membership. The document was approved by the members of the ATSC on 25 February 2003.

This document¹ defines the ATSC Transport Stream File System (TSFS) standard for delivery of hierarchical name-spaces, directories and files. This standard builds on the data service delivery mechanism defined in the ATSC Data Broadcast Standard [A90]. It was designed based on responses to an RFP whose requirements are listed in Annex C.

1.1 Organization

This document is organized as follows:

- **Section 1** — Provides this general introduction.
- **Section 2** — Lists references and applicable documents.
- **Section 3** — Provides a definition of compliance notation, acronyms, abbreviations, terms, syntax notations, and code points for this document.
- **Section 4** — Provides an informative description of the relationship of this standard to other standards and the overall architecture of the TSFS structures.
- **Section 5** — Describes the structures and object messages required by this standard.
- **Section 6** — Describes the usage of descriptors in the object messages defined in this standard.
- **Section 7** — Describes the binding of a TSFS to the ATSC transport.
- **Annex A** — Provides an informative discussion of TSFS design.
- **Annex B** — Provides an informative discussion of object acquisition in a TSFS.
- **Annex C** — Provides an informative description of the driving requirements for this standard, and how this standard meets them.

2. REFERENCES

2.1 Normative References

[DSMCC] ISO/IEC 13818-6:1998, “Information technology – Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC,” September 1, 1998, Sections 5.6.1, 5.6.3, 11.2.2-11.2.6, 11.3.1, 11.3.2.1-11.3.2.3, 11.3.2.5, 11.3.3.

¹ NOTE: The user’s attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of this claim, or of any patent rights in connection therewith. The patent holder has, however, filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Details may be obtained from the publisher.

- [MPEG] ISO/IEC 13818-1:2000, "Information technology – Generic coding of moving pictures and associated audio information: Systems," December 1, 2000.
- [SEC] ISO/IEC 16500-7:1999, "Information Technology – Generic digital audio-visual systems – Part 7: Basic security tools," December 16, 1999.
- [A90] ATSC Document A/90, "ATSC Data Broadcast Standard," July 26, 2000.
- [A65] ATSC Document A/65A, "ATSC Program and System Information Protocol for Terrestrial Broadcast and Cable", May 31, 2000.
- [MIME] IETF RFC 2045, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," section 5, November 1996.
- [URI] IETF RFC 2396, "Uniform Resource Identifiers: Generic Syntax," August 1998.
- [URI-LID] SMPTE 343M-2002, "Declarative Data Essence – Local Identifier (lid:) URI Scheme," 2000.
- [IEEE802] IEEE Std 802-1990, "IEEE Standards for Local and Metropolitan Networks: Overview and Architecture," May 31, 1990.
- [Trigger] ATSC Document A/93, "ATSC Synchronized/Asynchronous Trigger Standard," February 4, 2002.

2.2 Informative References

- [DVB-DB] ETSI EN 301 192 v1.2.1, "Digital Video Broadcasting (DVB); specification for data broadcasting," June 1999.
- [DVB-IG] ETSI TR 101 202 v1.1.1 (1999-02), "Digital Video Broadcasting (DVB); Implementation Guidelines for Data Broadcasting," February 1999.
- [DVB-MHP] ETSI TS 102 812 v1.1.1 (2001-11), "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP)," specification 1.1 (November 2001), Annex B (Normative): Object Carousel, section B.2.3.4.
- [SMPTE-EBU] SMPTE/EBU "Final Report of the Task Force for Harmonized Standards for the Exchange of Program Material and Bitstreams," July 1998.
- [MRD] ATSC Technology Group Report T3-548, "ATSC Usage of the MPEG-2 Registration Descriptor," October 9, 2001.
- [COLLISION] NTSC Technology Group Report T3-549, "Collision Avoidance for Private Fields and Ranges," October 9, 2001.
- [RFP] ATSC T3/S13 Document S13-148, "Request for Proposal for Potential Revisions to ATSC Standards in the Area of Transport Stream File System Functionality," March 28, 2001.

3. DEFINITIONS AND STRUCTURES

3.1 Compliance Notation

As used in this document, "shall" denotes a mandatory provision of the standard. "Should" denotes a provision that is recommended but not mandatory. "May" denotes a feature whose presence does not preclude compliance, that may or may not be present at the option of the implementer.

3.2 Acronyms and Abbreviations

The following acronyms and abbreviations are used within this specification:

AFI	authority and format identifier
BIOP	broadcast inter-ORB protocol
bslbf	bit serial, leftmost bit first
CRC	cyclic redundancy check
DAU	data access unit
DDB	download data block
DII	download information indication
DSI	download server initiate
DSM-CC	digital storage media command and control
DST	data service table
IOR	interoperable object reference
MPEG	Moving Picture Experts Group
NRT	network resources table
NSAP	network service access point
ORB	object request broker
OUI	organizationally unique identifier, as defined in [IEEE802]
PID	packet identifier
PMT	program map table
PSIP	Program and System Information Protocol
TS	transport stream
TSID	transport stream ID
TSFS	transport stream file system
uimsbf	unsigned integer, most significant bit first
URI	uniform resource identifier
U-U	user-to-user

3.3 Global Terms

The following terms are used throughout this document:

ATSC Advanced Television Systems Committee. The committee responsible for the coordination and development of voluntary technical standards for advanced television systems.

aligned A bit in a coded bit stream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.

binding A binding is a collection of bytes in a directory object that defines an association between a name and a reference (IOR) to an object. A binding may also contain descriptive information about the bound object.

binding name The name that appears in a binding.

binding structure The `DirectoryMessageBody()` of a directory object, consisting of a list of bindings associating object names with their locations.

BIOP object An object formatted according to the generic BIOP message structure defined in [DSMCC].

carousel A carousel identifies a group of objects transmitted repeatedly from a particular service provider for a specific purpose (service).

content type A content-type is the top-level media type used to declare the general type of data, as defined in [MIME]. A subtype is used to convey a specific format for that type of data. For example, a media type of “image/xyz” indicates that the data is an image, even without knowledge of the specific image format “xyz”.

CRC The cyclic redundancy check used to verify the correctness of the data.

data receiver Any device capable of receiving and consuming data carried in an ATSC transport stream.

data service A collection of scheduled applications and associated data elementary streams as signaled in one Data Service Table. A data service is characterized by a profile and level.

data source The provider of data that is being inserted into the MPEG-2 transport stream.

decoded stream The decoded reconstruction of a compressed bit stream.

decoder An embodiment of a decoding process.

decoding (process) The process defined in the Digital Television Standard that reads an input coded bit stream and outputs decoded pictures, audio samples, or data objects.

directory link An alternative term for a binding.

directory path A sequence of directory links, in which for each link in the sequence except the last one the object referenced by the link is the directory containing the next link in the sequence.

elementary stream (ES) A generic term for one of the coded video, coded audio, or other coded bit streams. One elementary stream is carried in a sequence of PES packets with one and only one `stream_id`.

forbidden This term, when used in clauses defining the coded bit stream, indicates that the value shall never be used. This is usually to avoid emulation of start codes.

interoperable object reference A data structure `IOP::IOR` that contains the information necessary to locate an object in a network; originally developed as part of the CORBA specification, later specialized by the ISO DSM-CC Standard to the case of an MPEG-2 broadcast network.

Kbps 1,000 bits per second.

lid A URI scheme defined by [URI-LID].

Mbps 1,000,000 bits per second.

module delivery parameters The delivery parameters of data modules are conveyed in `DownloadInfoIndication` messages. One `DownloadInfoIndication` message can convey the module delivery parameters of multiple data modules of the same U-U Object Carousel.

MPEG Refers to standards developed by the ISO/IEC JTC1/SC29 WG11, *Moving Picture Experts Group*. MPEG may also refer to the Group.

MPEG-2 Refers to the collection of ISO/IEC standards 13818-1 through 13818-6.

multiplexer (mux) A physical device that is capable of inserting MPEG-2 transport stream packets into and deleting MPEG-2 transport stream packets from an MPEG-2 transport stream.

NSAP address Network Service Access Point (NSAP) consists of AFI, Type, carouselId, specifier, privateData, as specified in Figure 11-2 of [DSM-CC]. It is a globally unique identifier that is used to identify a particular service domain.

object An object is an entity transmitted using the Object Carousel Protocol; it is a serialized object rather than an object definition. This could be raw data representing a file, a directory or a service gateway.

object key A collection of bytes that uniquely identifies an object of a TSFS within the data carousel module that contains it.

packet A packet is a set of contiguous bytes consisting of a header followed by its payload.

packet identifier (PID) A 13-bit identifier appearing in the header of an MPEG-2 Transport Stream packet that is used to associate Transport Stream packets of a program element or other data stream, such as a PSI or PSIP stream.

payload The bytes following the header bytes in a packet. The adaptation header of an MPEG-2 Transport Stream packet is not considered part of the payload of the packet.

program A collection of program elements. Program elements may be elementary streams. Program elements need not have any defined time base; those that do have a common time base and are intended for synchronized presentation. The term *program* is also commonly used in the context of a “television program” such as a news broadcast scheduled daily. In this specification the term “event” is used for the latter to avoid ambiguity.

program element A generic term for one of the elementary streams or other data streams that may be included in an ISO/IEC 13818-1 (MPEG-2) Program. The MPEG-2 Transport Stream packets conveying a Program Element are referenced by a unique PID value in the MPEG-2 Program.

program specific information (PSI) PSI consists of normative data defined in [MPEG] which is necessary for the demultiplexing of transport streams and the successful regeneration of programs.

profile A defined subset of data delivery characteristics. This differs from the definition provided in ISO/IEC 13818-2.

PSIP ATSC Program and System Information Protocol, which defines a collection of tables describing virtual channel attributes, event features, and other information.

reserved This term, when used in clauses defining the coded bit stream, indicates that the field may be used in the future for Digital Television Standard extensions. All reserved bits are set to ‘1’.

section A data structure comprising a portion of an ISO/IEC 13818-1 or ISO/IEC 13818-6-defined table, such as the Program Association Table (PAT), Conditional Access Table (CAT), Program Map Table (PMT), or DSMCC section. All sections begin with the table_id and end with a checksum or a CRC_32 field, and their starting points within a packet payload are indicated by the pointer_field mechanism defined in the ISO/IEC 13818-1 International Standard.

service description framework The information conveyed in the program element providing the Data Service Table and optionally the Network Resource Table of a single data service.

service domain A Service Domain identifies a structured group of objects. It is an abstract entity defined for the purpose of scoping. Each instance of an Object Carousel represents a Service Domain. Each Service Domain is identified by a globally unique NSAP address. Each Service Domain has a Service Gateway serving as its root directory.

service gateway A Service Gateway is the one and only entry-point to the content that is broadcast by the Object Carousel. It serves as the root directory of the Object Carousel. There is a Service Gateway associated with each Service Domain.

stream An ordered series of bytes. The usual context for the term *stream* is the series of bytes extracted from Transport Stream packet payloads which have a common unique PID value (e.g., video PES packets or Program Map Table sections).

table A collection of re-assembled sections bearing a common `table_id` and version number.

table instance A collection of re-assembled sections with a common `table_id`, `table_id_extension`, and `version_number`. Examples are the PSIP EITs and the Data Broadcasting DETs, where the `source_id` appears in the `table_id_extension` field to distinguish different instances of the tables.

tap A data structure used to establish a link from an object reference to a lower layer communication channel.

transport stream Refers to the MPEG-2 transport stream syntax for the packetization and multiplexing of video, audio, and data signals for digital broadcast systems [MPEG].

transport stream packet header The leading fields in an MPEG-2 Transport Stream packet up to and including the `continuity_counter` field.

URI Uniform Resource Identifiers (URIs) provide a simple, unified and extensible mechanism for identifying a resource. The URI framework [URI] provides a way to concatenate a base-URI with a relative-URI to form an absolute URI identifying a resource.

virtual channel number A virtual channel number is the designation that is recognized by the user as the single entity that will provide access to an analog TV programming or a set of one or more digital elementary streams. It is called “virtual” because its identification (name and number) may be defined independently from its physical location.

virtual channel A virtual channel is an analog TV broadcast or a set of one or more digital TV program elements providing a unified service and identified by a virtual channel number.

3.4 Section and Data Structure Syntax Notation

Tables defined in this standard conform to the generic private section syntax defined in [MPEG] and the DSM-CC section format defined in [DSMCC]. This document contains symbolic references to syntactic elements. The notation used is distinctive to aid the reader in recognizing elements that are the same as they are in referenced standards. These references are typographically distinguished by the use of a different font (e.g., *restricted*), may contain the underscore character (e.g., `sequence_end_code`) and may consist of character strings that are not English words (e.g., `dynrng`).

reserved — Fields in this Standard marked “reserved” shall not be assigned by the user, but shall be available for future use. Decoders are expected to disregard reserved fields for which no

known definition exists for the decoder. Each bit in the fields marked “reserved” shall be set to ‘1’ until such time as they are defined and supported.

user_private — A value or range of values of a code point that may be privately defined by users of a particular standard. It must be possible to determine the identity of the standards body or private party specifying a user private value. In instances where the ownership of the definition of the bytes is not otherwise explicit, the MPEG-2 Registration Descriptor shall be used for this purpose.

zero — Indicates that the bit or bit field shall have the value zero.

3.5 Elements with Undefined Semantics

In a number of places in this standard there appear statements that certain elements (which have been included for compatibility with [DSMCC] and [DVB-MHP]) “have no meaning in this standard.” This means that such elements will normally not be present in Transport Stream File Systems implementing this standard, but implementers should not assume that such elements will never be present. It is possible that there may be a future version of this standard in which some of these elements are used in order to support some kind of extended functionality.

3.6 Code Points (Informative)

For convenience, this section lists all values of `table_id`, `stream_type`, `descriptor_tag`, `encapsulation_protocol`, and `selector_type` that are used in this standard.

3.6.1 Table ID Values

The `table_id` values defined or used in this standard are:

- 0x3B DSM-CC section containing DSI or DII message (defined in [DSMCC]).
- 0x3C DSM-CC section containing DDB message (defined in [DSMCC]).

3.6.2 Stream Type Values

The `stream_type` value used in this standard is:

- 0x0B DSM-CC sections containing data carousel messages (defined in [DSMCC]).

3.6.3 Descriptor Tag Values

The `descriptor_tag` values used in this standard are:

- 0x72 Content Type descriptor (defined in [Trigger]).
- 0xB9 Time Stamp descriptor (defined in this standard).

3.6.4 Encapsulation Protocol Values

The `encapsulation_protocol` value defined in this standard is:

- 0x0F Transport Stream File System.

3.6.5 Selector Type Values

The `selector_type` values defined or used in this standard are:

- 0x0001 Message selector (defined in [DSM-CC]).
- 0x109 TSFS selector (defined in this standard).

4. INTRODUCTION (INFORMATIVE)

This section describes the overall architecture of the ATSC Transport Stream File System. The ATSC Transport Stream File System is based on the Object Carousel design specified in [DSMCC], section 11. Figure 4 shows how this standard fits in with other standards.

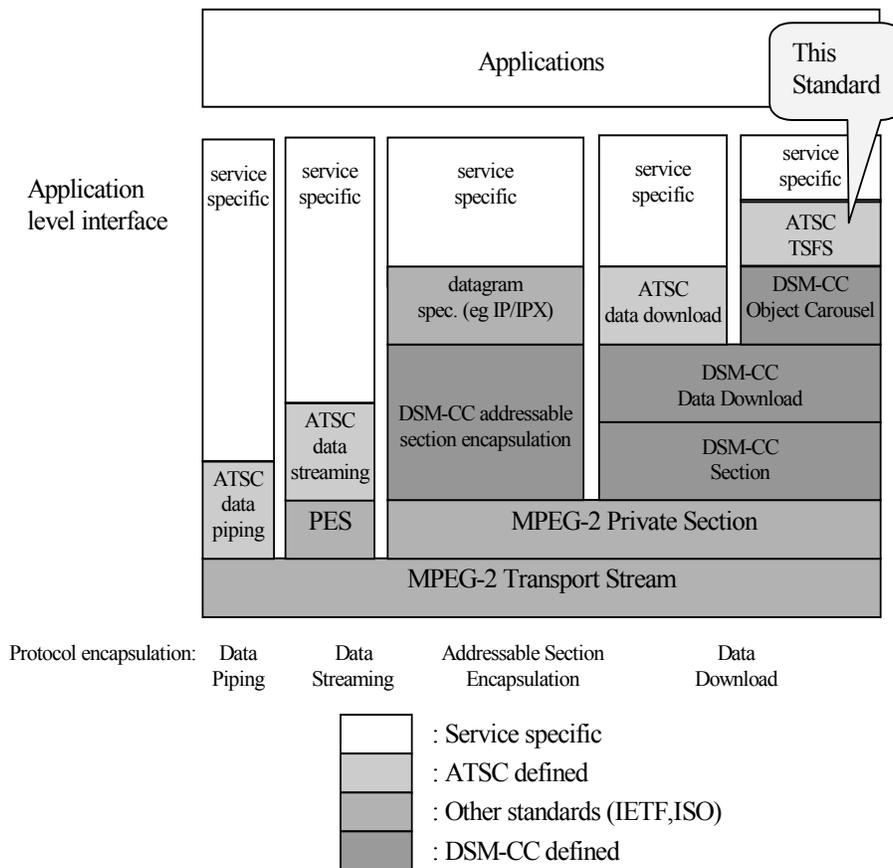


Figure 4.1 Graphical encapsulation overview and relation to other standards.

The ATSC TSFS Standard supports the transmission of the following types of BIOP objects defined in section 11.3.2 of [DSMCC]: DSM::Directory, DSM::File, and DSM::ServiceGateway. Carriage of these objects is done by means of the DSM-CC Data Carousel protocol as described in sections 11 and 9 of [DSMCC]. However, the ATSC TSFS standard further constrains the syntax and semantics of the BIOP objects to satisfy the requirements set forth in [RFP].

Each of these objects contains a header giving the object key (unique identifier within the data carousel module containing it) and the type (directory, file, or service gateway) of the object. The header of a DSM::File object also has an objectInfo structure giving the file size, content-type (MIME type), and a “last modified” time stamp. The header of a DSM::Directory or DSM::ServiceGateway object has an objectInfo structure that optionally contains a “last modified” time stamp. It does not convey a content-type, since a content-type as defined in [MIME] is not applicable to directory objects. The body of a DSM::File object is just the file content. The body of

a DSM::ServiceGateway or DSM::Directory object consists of a list of bindings (references to other objects, with associated binding names).

A binding contains a binding name, the object type of the referenced object (directory, file, or service gateway), an IOP::IOR (Interoperable Object Reference) giving the location of the referenced object, and an objectInfo structure that may contain some or all of the same information as the objectInfo structure in the header of the referenced object. A binding is often called a directory link or a directory entry, and an IOP::IOR is often just called an IOR.

Figure 4.2 illustrates the general structure of a DSM::Directory object and DSM::File object. A DSM::Directory object includes an objectInfo structure and a binding structure referencing zero or more child objects (DSM::Directory or DSM::File objects). The binding information includes an objectInfo field for the purpose of providing early information about a referenced object. When descriptors are listed in such an objectInfo structure, their contents are identical to the contents of the descriptors listed in the objectInfo structure that appears in the header of the object itself.

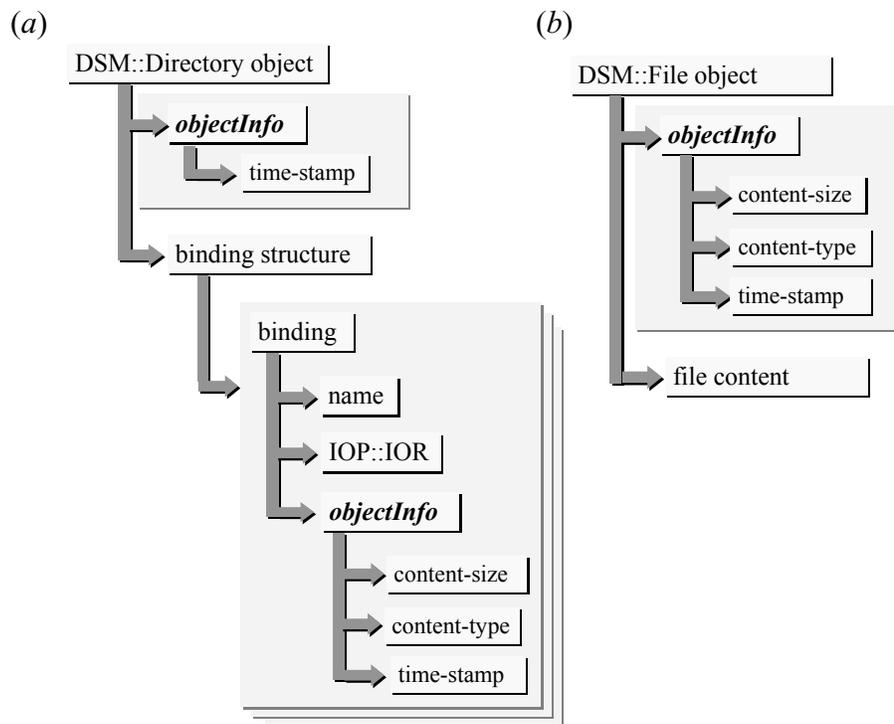


Figure 4.2 (a) Directory object (b) File object.

A DSM::ServiceGateway object is similar to a DSM::Directory object. The main difference between these two objects is simply in the syntax of the name or names listed in the binding structure. For a DSM::ServiceGateway object, a binding name is always a base URI while for a DSM::Directory object a binding name is a relative path.

Each TSFS has exactly one Service Gateway object, which serves as the top level directory of the TSFS. The objects in the TSFS can all be reached by following a directory path (sequence of directory links) starting from the Service Gateway. An object in a TSFS can be referenced from multiple directories in the TSFS, so there can be multiple paths from the Service Gateway to an object in the TSFS. Such paths cannot create any circular paths in the directory reference

structure. For example, the situation where directory A contains a reference to directory B, which in turn contains a reference to directory A, cannot occur.

As mentioned above, the names that appear in the binding structures of the Service Gateway object all conform to the syntax of an absolute URI, as specified in [URI], and the names that appear in the binding structures of lower level directories of a TSFS all conform to the syntax of a relative path, as specified in [URI]. The effect of this is that each file or directory object in a TSFS, with the exception of the Service Gateway object itself, has an associated absolute URI that can be used to reference the object, obtained by concatenating the names in a sequence of directory references leading from the Service Gateway to the object, with slashes (/) as delimiters between them. If there are multiple such sequences of directory references leading to the object, then it has multiple URIs associated with it.

An IOR referencing an object in the same TSFS contains the object key for the referenced object, the carousel ID and module ID of the data carousel module containing the object, a Tap identifying the Program Element containing the DownloadInfoIndication (DII) message describing the delivery parameters for the module, and optionally a Tap identifying the Program Element containing the DownloadDataBlock (DDB) messages carrying the module itself. The DII message gives such information as the block size, module size, module version, and delivery timeout intervals for the module, together with a Tap identifying the Program Element containing the DDB messages carrying the module. All objects of a TSFS are carried in a single virtual channel. However, it is also possible to have bindings that contain references to objects in other TSFSs (often called “soft links”). These other TSFSs may be in the same virtual channel or a different virtual channel, even a different virtual channel in a different transport stream. Thus, the logical name space of a TSFS may span multiple virtual channels and even multiple transport streams.

An IOR referencing an object in a different TSFS identifies the TSFS containing the object and gives the directory path (sequence of directory links) in that TSFS that leads from the Service Gateway of that TSFS to the object.

Organization of the ATSC TSFS into the Data Modules of an ATSC Data Carousel [A90] is not specified by this standard. For example, the BIOP Service Gateway and BIOP Directory objects can be located in one or multiple data modules that may or may not be separate from the data modules conveying the BIOP File objects. Such a design can be used to allow more frequent transmissions of the data modules conveying the BIOP Service Gateway and the BIOP Directory objects so upon tuning to a new Virtual Channel, receivers get an opportunity to reconstruct an image of the file hierarchy with minimum latency.

There is a constraint imposed by [DSMCC] (in Sections 7.5.4 and 9.2.5) that all the DDB messages carrying the modules of a single data carousel must be in the same program element, and DDB messages carrying modules of different data carousels must be in different program elements. However, the DSI and DII messages of a data carousel can be in the same program elements as the DDB messages or can be in one or more other program elements, and the DSI and DII messages of different data carousels can appear in the same program element.

To allow unambiguous identification of a TSFS in an ATSC Transport Stream, a new `protocol_encapsulation` value is defined for the Data Service Table (DST) of the Data Broadcast Standard [A90] to signal the Object Carousel encapsulation. For each TSFS used by an application in a data service, the DST contains a Tap referencing the Program Element containing the DSI message that signals the location of the Service Gateway object for the TSFS.

In the case where this Program Element resides in a remote ATSC Virtual Channel or a remote ATSC Transport Stream, it is signaled in the Network Resources Table (NRT).

5. STRUCTURES

5.1 Carousel NSAP Address

Each instance of a TSFS represents a Service Domain, which is defined in Section 1.1 of [DSMCC] as “a collection of interfaces to browse and select services” and is later referred to informally in Section 5.1.2 of [DSMCC] as a “directory hierarchy.” Each Service Domain shall have a globally unique identifier associated with it, called the Carousel NSAP (Network Service Access Point) address. Table 5.1 describes the Carousel NSAP address structure, with overall syntax as specified in [DSMCC] and `privateData()` syntax as specified by this standard.

Table 5.1 Carousel NSAP Address Syntax

Syntax	No. of Bits	Format
CarouselNSAPaddress() {		
AFI	8	uimsbf
type	8	uimsbf
carouselId	32	uimsbf
specifierType	8	uimsbf
specifierData	24	uimsbf
privateData() {		
transportStreamID	16	uimsbf
originalTSID	16	uimsbf
program_number	16	uimsbf
sourceId	16	uimsbf
originalSourceId	16	uimsbf
}		
}		

AFI — This 8-bit Authority and Format Identifier field shall be set to 0x00, as specified in Section 11.2.2 of [DSMCC].

type — This 8-bit field shall be set to 0x00 indicating that the Carousel NSAP address points to a U-U Object Carousel. The values in the range 0x01 to 0x7F are reserved to ISO/IEC 13818-6. The values in the range 0x80 to 0xFF are user private and their use is outside the scope of this specification.

carouselId — This 32-bit field shall uniquely identify the carousel within the Virtual Channel.

specifierType — This 8-bit field shall be set to 0x01 to indicate that the `specifierData` field is an OUI, as specified in [DSMCC] Chapter 6.

specifierData — This 24-bit field shall be set to the IEEE OUI of ATSC, 0x000979. This field, along with the `specifierType`, uniquely scopes the combination of `transportStreamId` and `source_id` fields that follow.

transportStreamID – This 16-bit field shall contain the `transport_stream_id` (TSID) of the MPEG-2 transport stream containing the TSFS, as it appears in the Program Association Table of the transport stream. During remultiplexing it may be changed to reflect the TSID of the new transport stream carrying the TSFS.

originalTSID – This 16-bit field shall contain the TSID of the MPEG-2 transport stream in which the virtual channel containing this TSFS was originally broadcast. It shall not be changed during remultiplexing operations.

Note: Coordination of sources to ensure uniqueness of `originalTSID` is outside the scope of this standard.

program_number – This 16-bit field shall contain the `program_number` from the `program_map_section` [MPEG] associated with the Virtual Channel conveying the Service Gateway object of this TSFS.

source_id – This 16-bit field shall contain the `source_id` for the virtual channel carrying the TSFS, as it appears in the Virtual Channel Table. (Note that a TSFS may appear in more than one virtual channel, since virtual channels may have overlapping program elements, so that a TSFS may have more than one NSAP address associated with it. However, a single NSAP address cannot refer to more than one TSFS.) During re-multiplexing it may be changed to reflect a new `source_id` for the virtual channel.

originalSourceId – This 16-bit field shall contain the `source_id` for the virtual channel in which this TSFS was originally broadcast. It shall not be changed during re-multiplexing operations.

Note that if the `source_id` is in the range 0x1000 to 0xFFFF, so that it is unique at the regional level (per [A65]), then the `source_id` will never need to change during typical remultiplexing (e.g., for cable carriage), and a receiver will normally be able to find the correct virtual channel containing the TSFS with a given Carousel NSAP Address from the `source_id` alone. If the `source_id` is in the range from 0x0001 to 0x0FFF, then the combination of `originalTSID` and `originalSourceId` forms an identifier for the virtual channel that is unique and invariant under remultiplexing, but it will take more effort for a receiver to actually locate the TSFS from this identifier. A discussion of the resolution process for a Carousel NSAP address appears in Annex B Section 3.

5.2 Interoperable Object Reference (IOR) Format

An ATSC Transport Stream File System is a collection of DSM-CC BIOP object structures of types `DSM::Directory`, `DSM::File`, and `DSM::ServiceGateway`. These objects are organized into a directory hierarchy by means of bindings in the Directory and Service Gateway objects that associate names with references to objects.

References to objects in an ATSC TSFS shall follow the format of an IOR, as specified in Sections 5.6 and 11.3.1 of [DSMCC], with the additional constraints specified in this section. Table 5.2 presents the constrained IOR structure. Table 5.3 shows the values of the `type_id` field corresponding to each type of BIOP object that may be referenced by an IOR according to this standard. Other object types have no meaning in this standard.

Table 5.2 Definition of the IOR Structure

Syntax	No. of Bits	Format
IOP::IOR () {		
typeld_length	32	uimsbf
for (i=0; i<typeld_length; i++) {		
typeld_byte	8	bslbf
}		
taggedProfile_count	32	uimsbf
IOP::TaggedProfile() {		
BIOPProfileBody() or LiteOptionsProfileBody()	variable	bslbf
}		
for (n=0, n<taggedProfile_count-1; n++) {		
IOP::TaggedProfile()	variable	bslbf
}		
}		

typeld_length — This 32-bit field shall be set to 0x00000004 indicating the use of the 3-character alias with a terminating null byte.

typeld_byte — This 8-bit field shall contain one byte of the object type as specified in Table 5.3.

Table 5.3 Value of typeld_bytes

typeld_bytes	Description
0x64697200 (ASCII "dir" with null terminator)	DSM::Directory object
0x66696C00 (ASCII "fil" with null terminator)	DSM::File object
0x73726700 (ASCII "srg" with null terminator)	DSM::ServiceGateway

Note that in each case the value of typeld_byte includes a null byte terminator for the ASCII string it represents (i.e., a byte with value 0x00).

taggedProfile_count — This 32-bit field shall be set to the number of IOP::TaggedProfile structures that are present. Any instances of IOP::TaggedProfile that appear after the first one have no meaning in this standard.

Complete descriptions of the BIOPProfileBody and LiteOptionsProfileBody appear in Sections 5.3 and 5.4, respectively.

5.3 References within the Same TSFS

When an IOR is used to reference a BIOP object within the same TSFS, the IOR shall contain a BIOPProfileBody, as defined in Sections 11.3.1.1, 5.6.1, and 5.6.3.5 of [DSMCC] and shown in Table 5.4. All objects within a single TSFS shall be conveyed in program elements that are within the same MPEG-2 program. The BIOPProfileBody presented in Table 5.4 is a constrained version of the BIOPProfileBody structure defined in [DSMCC]. It carries all the information pertaining to an object that is needed to uniquely identify the object and locate it within a Service

Domain (specified by an NSAP address). The first two components of the `BIOPProfileBody` shall be `BIOP::ObjectLocation` and `DSM::ConnBinder` components, in that order. Any additional components have no meaning in this standard.

The `BIOP::ObjectLocation` component provides the 3-tuple `<carouselId, moduleId, ObjectKey>`, which identifies the object uniquely. The `DSM::ConnBinder` structure uses the `DSM::Tap()` structure defined in Section 5.6.1 of [DSMCC] to indicate where the object can be found in the MPEG-2 program. The first `Tap` in the `DSM::ConnBinder` structure of the `BIOPProfileBody` shall identify the Program Element that has the DII message giving the delivery parameters for the module containing the object referenced by the IOR. There may be an additional `Tap` which identifies a Program Element that has the DDB messages carrying the module itself. Any further taps in the `DSM::ConnBinder` structure have no meaning in this standard.

Table 5.4 BIOPProfileBody Structure

Syntax	No. of Bits	Format
BIOPProfileBody {		
profileId_tag	32	uimsbf
profile_data_length	32	uimsbf
profile_data_byte_order	8	uimsbf
component_count	8	uimsbf
BIOP::ObjectLocation {		
objectLocation_tag	32	uimsbf
objectLocation_length	8	uimsbf
carouselId	32	uimsbf
moduleId	16	uimsbf
version.major	8	uimsbf
version.minor	8	uimsbf
objectKey {		
objectKey_length	8	uimsbf
For (k=0; k< objectKey_length; k++) {		
objectKey_byte	8	bslbf
}		
}		
}		
DSM::ConnBinder {		
connBinder_tag	32	uimsbf
connBinder_length	8	uimsbf
tap_count	8	uimsbf
for (j=0; j<tap_count; j++) {		
DSM::Tap {		
tapId	16	uimsbf
tapUse	16	uimsbf
associationTag	16	uimsbf
selector()	variable	bslbf
}		
}		
}		
for (j=0; j<component_count - 2; j++) {		
LiteOptionsComponent()	variable	bslbf
}		
}		

profileId_tag — This 32-bit field shall be set to 0x49534F06 indicating that this is a BIOP Protocol Profile (see [DSM-CC] Section 5.6.3.5).

profile_data_length — This 32-bit field shall specify the total number of bytes in this BIOPProfileBody structure following (i.e., excluding) this field.

profile_data_byte_order — This 8-bit field shall be set to 0x00 (FALSE) indicating big-endian byte ordering for the subsequent fields of the message.

component_count — This 8-bit field shall be set to the number of components in this BIOPProfileBody structure. The first component shall be BIOP::ObjectLocation and the second shall be DSM::ConnBinder. Any further components have no meaning in this standard.

objectLocation_tag — This 32-bit field shall be set to 0x49534F50 as specified in Section 5.6.3.5 of [DSMCC], indicating that this is a BIOP::ObjectLocation structure.

objectLocation_length — This 8-bit field shall specify the number of bytes in the BIOP::ObjectLocation structure following this field (not including this field).

carouselId — This 32-bit field shall be set to the downloadId value for the download scenario of the module containing the object, as specified in Section 11.3.3.1 of [DSMCC]. All objects of a single TSFS shall be carried in modules of the same download scenario. All DownloadDataBlock and DownloadInfoIndication messages within the download scenario share the same downloadId, as specified in Section 11.3.3.2 of [DSMCC].

moduleId — This 16-bit field shall be set to the moduleId of the module containing the object, as defined in [DSM-CC]. The moduleId values in the range 0xFFFF0-0xFFFF are reserved per [SEC].

version.major — This 8-bit field shall contain the value 0x01, indicating the major version number of the BIOP protocol used in this message.

version.minor — This 8-bit field shall contain the value 0x00, indicating the minor version number of the BIOP protocol used in this message.

objectKey_length — This 8-bit field shall specify the number of bytes in the object key. The number of bytes shall be no greater than 4 bytes, so as to avoid large keys. Note that object keys are only intended to identify the objects uniquely within a specific module of a specific TSFS and are not intended to serve as the URIs of the objects.

objectKey_byte — This 8-bit field shall contain a byte of the object key, which uniquely identifies the object within the module containing the object. Two objects in the same module are regarded as equivalent if and only if their objectKey_byte fields are identical.

connBinder_tag — This 32-bit field shall be set to 0x49534F40 indicating that this is a DSM::ConnBinder structure, as specified in Section 5.6.3.3 of [DSMCC].

connBinder_length — This 8-bit field shall be set to the total length in bytes of this DSM::ConnBinder structure after this field (not including this field).

tap_count — This 8-bit field shall indicate the number of Taps in this DSM::ConnBinder structure.

Note: The DSM::Tap() structure used here is defined in Section 5.6.1 of [DSMCC].

tapId — The value of this 16-bit field shall be set to 0xFFFF, indicating that the usage of this field is reserved.

tapUse — The first Tap in the list of Taps shall have this 16-bit field set to 0x0016 (BIOP_DELIVERY_PARA_USE), indicating that this Tap identifies a Program Element containing a DII message describing the delivery parameters for the module containing the object. There may be another Tap in the list that has this 16-bit field set to 0x0017 (BIOP_OBJECT_USE), indicating that the Tap identifies a Program Element containing the DDB messages conveying the module itself. (The semantics of these values of tapUse are specified in [DSMCC], Section 11.3.1.1.2.) Any Taps with other values of tapUse have no meaning in this standard.

associationTag — This 16-bit field shall uniquely identify a Program Element listed in the Program Map Section for the current virtual channel. The value of this field shall match the association_tag value of an association_tag_descriptor in the Program Map Section.

selector() — The selector() structure is defined in Section 5.6.1 of [DSMCC] and shown in Table 12.8 of [A90]. When the value of tapUse is 0x0016 (BIOP_DELIVERY_PARA_USE), the selector() field shall contain a MessageSelector(), as defined in Section 5.6.1.1 of [DSMCC] and described in Table 5.5. This MessageSelector() structure is an instantiation of the generic selector() structure. It is used to specify which of the possibly multiple DII messages in the Program Element identified by a Tap is being referenced by the Tap. When the value of tapUse is 0x0017 (BIOP_OBJECT_USE), the selector() shall consist solely of an 8-bit selector_length field, which shall be set to 0x00 to indicate that the remainder of the selector() is empty.

Table 5.5 MessageSelector Structure

Syntax	No. of Bits	Format
MessageSelector() {		
selector_length	8	uimsbf
selector_type	16	uimsbf
transactionId	32	uimsbf
timeout	32	uimsbf
}		

selector_length — This 8-bit field shall be set to 0x0A, indicating that the number of bytes in the selector() structure following this field (not including this field) is 10.

selector_type — This 16-bit field shall be set to 0x0001 to indicate that the selector() is a MessageSelector().

transactionId — This 32-bit field shall be set to the transactionId of the DII message to be referenced.

timeout — This 32-bit field shall indicate the timeout period in microseconds to be used to time out the acquisition of the DII message.

5.4 References to Remote Objects

When an IOR is used to reference a BIOP object residing in a different TSFS from that in which the IOR appears, the IOR shall contain a LiteOptionsProfileBody containing a ServiceLocation component as defined in Section 5.4.2 of [DSMCC] and described in Table 5.6. The Service Location component contains the NSAP address of the other TSFS and a Name() structure

consisting of a list of names and corresponding object types, providing a specification for the directory path from the Service Gateway of that TSFS to the referenced object. Table 5.1 describes the syntax of the Carousel NSAP Address. Annex B Section 3 describes how the Carousel NSAP Address of a LiteOptionsProfileBody can be resolved to find the corresponding Service Gateway of the remote TSFS.

Table 5.6 Definition of the LiteOptionsProfileBody Structure

Syntax	No. of Bits	Format
LiteOptionsProfileBody {		
profileId_tag	32	uimbsf
profile_data_length	32	uimbsf
profile_data_byte_order	8	uimbsf
component_count	8	uimbsf
DSM::ServiceLocation {		
componentId_tag	32	uimbsf
component_data_length	8	uimbsf
carouselNSAPaddress_length	8	uimbsf
carouselNSAPaddress()	160	
Name() {		
nameComponents_count	32	uimbsf
for (j=0; j<nameComponents_count; j++) {		
nameComponent_length	32	uimbsf
for (k=0; k<nameComponent_length; k++) {		
nameComponent_data	8	bslbf
}		
kind_length	32	uimbsf
kind_data	32	uimbsf
}		
InitialContext_length	32	uimbsf
for (j=0; j<initialContext_length; j++) {		
InitialContext_data_byte	8	bslbf
}		
}		
for (n=0; n<component_count-1; n++) {		
LiteComponent()		
}		
}		

profileId_tag — This 32-bit field shall be set to 0x49534F05 indicating that this is a Lite Options Protocol Profile (see [DSM-CC] Section 5.6.3.4).

- profile_data_length** — This 32-bit field shall specify the total number of bytes in this LiteOptionsProfileBody structure following (i.e., excluding) this field.
- profile_data_byte_order** — This 8-bit field shall be set to 0x00 (FALSE) indicating big-endian byte ordering for the subsequent fields of the message.
- component_count** — This 8-bit field shall be set to the number of components that appear in this LiteOptionsProfileBody. The first component shall be a Service Location structure. Any other components that may be present have no meaning in this standard.
- componentId_tag** — This 32-bit field shall be set to 0x49534F46, indicating that this is a Service Location structure.
- component_data_length** — This 8-bit field shall specify the number of bytes in the DSM::ServiceLocation structure following this field (not including this field).
- carouselNSAPAddress_length** — This 8-bit field shall be set to 0x14, indicating that the Carousel NSAP Address is 20 bytes long.
- CarouselNSAPAddress()** — This 20-byte field shall contain the Carousel NSAP Address of the remote TSFS.
- nameComponents_count** — This 8-bit field shall be set to the number of directory links in the directory path from the Service Gateway to the referenced object in the remote TSFS.
- nameComponent_length** — This 32-bit field shall be set to the length of the binding name, including the null terminating byte, for this link in the directory path from the Service Gateway to the referenced object in the remote TSFS.
- nameComponent_data** — This 8-bit field shall contain a byte of the binding name for this link in the directory path from the Service Gateway to the referenced object in the remote TSFS. The binding name shall be terminated by a null byte.
- kind_length** — This 32-bit field shall be set to 0x04 indicating the use of the 3-character alias with a null byte terminator for the object type of the bound object for this link in the directory path.
- kind_data** — This 4-byte field shall contain the object type of the bound object for this link, as specified in Table 5.7. Note that the object type of the last link in the path description identifies the object type of the object referenced by the IOR.

Table 5.7 Value of kind_data Bytes

typeid_byte	Description
0x64697200 (ASCII "dir" with null terminator)	DSM::Directory object
0x66696C00 (ASCII "fil" with null terminator)	DSM::File object

- InitialContext_length** — This 32-bit field shall be set to the length of the Initial Context data. Any initial context data that are present have no meaning in this standard.
- InitialContext_data_byte** — If InitialContext_length is non-zero, this 8-bit field shall contain a byte of Initial Context data. Such data have no meaning in this standard.

As an alternative to providing a name and object type in the DSM::ServiceLocation() for each directory link in the path from the remote Service Gateway to the referenced object, the nameComponents_count may be set to 0x0001, and a single name may be provided, with

nameComponent_data bytes containing the full path from the remote Service Gateway to the referenced object, obtained by concatenating the individual names in the path with a slash ('/') used as a delimiter, and with kind_data set to the object type of the referenced object.

Under the rules for link names given in Sections 5.5.1 and 5.5.3, and the rules for associating URIs with TSFS objects given in Section 5.6, this amounts to giving the URI of the referenced object as defined by its position in the remote TSFS. The object may then be located using the process described in Annex B Section 5.

5.5 BIOP Message Formats

An ATSC Transport Stream File System shall consist of DSM-CC BIOP objects, each of which shall be of type DSM::Directory, DSM::File, or DSM::ServiceGateway. Each TSFS shall contain exactly one object of type DSM::ServiceGateway. Every other object in the TSFS shall be reachable by at least one directory path from the Service Gateway object, i.e., by at least one sequence of bindings with the property that the first binding in the sequence is contained in the Service Gateway object, the object referenced by each binding in the sequence except the last one is a directory containing the next binding in the sequence, and the object referenced by the last binding in the sequence is the object to be reached. The references in such a sequence shall be in the form of Interoperable Object References (IORs) containing a BIOPProfileBody.

A TSFS may also contain bindings referencing objects in other Transport Stream File Systems in the same virtual channel or other virtual channels, possibly in other transport streams, through IORs containing a BIOPLiteOptionsProfileBody. This enables a TSFS name hierarchy to span multiple virtual channels logically, even though the TSFS is physically carried in a single virtual channel.

The TSFS objects are carried in BIOP object messages. These messages are instantiated from the Generic Message Format specified in [DSMCC]. Each message consists of a header, a sub-header, and a message body.

5.5.1 Directory Message Format

An object of type DSM::Directory shall be conveyed by a BIOP::DirectoryMessage, as defined in Section 11.3.2.2 of [DSMCC] and further constrained by this standard. The BIOP::DirectoryMessage is an instantiation of the Generic Object Message defined in [DSMCC]. The following rules constrain this instantiation:

- The objectKind field shall contain the value 0x64697200 (the null-terminated ASCII string "dir").
- The messageBody structure shall contain the BIOP::DirectoryMessageBody structure.
- The BIOP::DirectoryMessageBody structure consists of a collection of zero or more bindings. A binding correlates a binding name to an IOR and provides additional information about the object. The IOR shall include the BIOPProfileBody when the referenced object is in the same TSFS, and shall include the LiteOptionsProfileBody when the referenced object is in a different TSFS.

The semantics of the fields of the BIOP::DirectoryMessageBody are as defined in [DSM-CC] with the following additional constraints imposed by this ATSC TSFS Standard:

- The BIOP::Name() field contains the binding name of the referenced object. This shall conform to the syntax of a relative path with respect to the parent directory, as defined in [URI].

Characters which are not present in US-ASCII, as well as reserved characters, shall be encoded by first taking the UTF-8 representation of these characters, and then escaping the octets of the UTF-8 representation using the %xx escape syntax defined by [URI] Section 2.1. For example, the relative URI component “café” would be encoded as “caf%c3%a9”, since the character “é” is encoded in UTF-8 as the octet sequence 0xC3 0xA9.

- The objectInfo field in a binding may contain objectInfo descriptors for the bound object if those same descriptors are present in the objectInfo of the corresponding object message. The objectInfo descriptors are defined in Section 6.
- The IOR in a binding shall not reference a Service Gateway.
- There shall be no circular paths in the directory structure of a TSFS.; i.e., it shall not be possible to follow a sequence of directory links from object to object and arrive back at the starting point of the sequence. This shall be true whether the links are to local or remote objects.

Table 5.8 defines the syntax of the DSM::DirectoryMessage, as constrained by this standard.

Table 5.8 Syntax of Directory Message Format

Syntax	No. of Bits	Format
BIOP::DirectoryMessage() {		
MessageHeader() {		
Magic	32	uimsbf
biop_version.major	8	uimsbf
biop_version.minor	8	uimsbf
byte_order	8	uimsbf
message_type	8	uimsbf
message_size	32	uimsbf
}		
MessageSubHeader () {		
objectKey {		
objectKey_length	8	uimsbf
for (i=0; i<objectKey_length; i++) {		
objectKey_data_byte	8	bslbf
}		
}		
dirObjectKind_length	32	
dirObjectKind_data	32	
dirObjectInfo_length	16	uimsbf
dirObjectInfo {		
for (j=0; j<dirObjectInfo_length; j++) {		
dirObjectInfo_descriptor_byte	8	bslbf
}		
}		
serviceContextList_count	8	uimsbf

for (j=0; j<serviceContextList_count; j++) {		
serviceContext() {		
context_id	32	uimsbf
context_data_length	16	uimsbf
for (k=0; k<context_data_length; k++) {		
context_data_byte	8	bslbf
}		
}		
}		
messageBody_length	32	uimsbf
DirectoryMessageBody() {		
bindings_count	16	uimsbf
for (i=0; i<bindings_count; i++) {		
BIOP::Name() {		
nameComponent_count	8	uimsbf
id_length	8	uimsbf
for (j=0; j<id_length; j++) {		
id_byte	8	bslbf
}		
kind_length	8	0x04
kind_data	32	uimsbf
}		
binding_type	8	uimsbf
IOP::IOR()		
childObjectInfo_length	16	uimsbf
childObjectInfo {		
if (kind_data == 0x66696C00) {		
ContentSize	64	uimsbf
for (j=0; j<childObjectInfo_length - 8; j++) {		
childObjectInfo_descriptor_byte	8	bslbf
}		
}		
else {		
for (j=0; j<childObjectInfo_length; j++) {		
childObjectInfo_descriptor_byte	8	bslbf
}		
}		
}		
}		
}		

- magic** — This field shall be set to 0x42494F50 to indicate that this is a BIOP message. (The value of this field corresponds to the string “BIOP” encoded in ISO Latin-1.)
- biop_version.major** — This 8-bit field shall be set to 0x01, indicating the major version number of the BIOP protocol used in this message.
- biop_version.minor** — This 8-bit field shall be set to 0x00, indicating the minor version number of the BIOP protocol used in this message.
- byte_order** — This 8-bit field shall be set to 0x00 (FALSE) indicating big-endian byte ordering for the subsequent fields of the message (including `message_size`).
- message_type** — This 8-bit field shall be set to 0x00 to indicate a BIOP object message.
- message_size** — This 32-bit field shall contain the length in bytes of the `BIOP::DirectoryMessage`, including all bytes following this field (not including this field) up to the end of the `BIOP::DirectoryMessage`.
- objectKey_length** — This 8-bit field shall contain the length in bytes of the object key, which shall be no greater than 4 bytes.
- objectKey_data_byte** — This 8-bit field contains a byte of the object key. The object key shall identify the object that is conveyed in this message uniquely within the data carousel module that contains this message. The value of the object key is only meaningful to the Broadcast Server and is treated as an opaque identifier (i.e., is not interpreted) by the Client.
- dirObjectKind_length** — This 32-bit field shall be set to 0x00000004, indicating that the `dirObjectKind_data` field is 4 bytes long.
- dirObjectKind_data** — This 32-bit field shall be set to 0x64697200 (null-terminated ASCII “dir”), indicating that this is a directory object. It is identical to the Kind string that is present in all IORs referring to this object.
- dirObjectInfo_length** — This 16-bit field shall specify the length in bytes of the `objectInfo` structure for this directory object.
- dirObjectInfo_descriptor_byte** — This 8-bit field shall contain a byte of the `objectInfo` structure, which shall consist of a list of descriptors. See Section 6 for syntax and semantics of the descriptors that may appear in this structure.
- serviceContextList_count** — This 8-bit field shall be set to the number of Service Context elements appearing in the Service Context List. Any Service Context elements that appear have no meaning in this standard.
- context_id** — This 32-bit field contains an identifier for the Service Context element.
- context_data_length** — This 16-bit field shall contain the length of the Service Context data.
- context_data_byte** — This 8-bit field contains a byte of Service Context data.
- messageBody_length** — This 32-bit field shall contain the length in bytes of the `DirectoryMessageBody`, which consists of all bytes following this field (not including this field) up to the end of the `BIOP::DirectoryMessage`.
- binding_count** — This 16-bit field shall contain the number of bindings in this directory. For each binding, a `BIOP::Name` structure is defined, followed by an IOR, optionally followed by the `objectInfo` for the referenced object.
- nameComponent_count** — This 8-bit field shall be set to 0x01 indicating that the `Name()` structure for each binding consists of a single name component.

id_length — This 8-bit field shall specify the length in bytes of the identifier (name) for this binding, including the null byte terminator for the string.

id_byte — This 8-bit field shall contain a byte of the identifier (name) for this binding. The identifier shall conform to the syntax of a relative path, as defined in [URI], followed by a null byte terminator. Characters which are not present in US-ASCII, as well as reserved characters, shall be encoded by first taking the UTF-8 representation of these characters, and then escaping the octets of the UTF-8 representation using the %xx escape syntax defined by [URI] Section 2.1. For example, the relative URI component “café” would be encoded as “caf%c3%a9”, since the character “é” is encoded in UTF-8 as the octet sequence 0xC3 0xA9. All binding names in a single directory object shall be distinct.

kind_length — This 8-bit field shall be set to 0x04 to indicate that the object type is 4 bytes (32 bits) long.

kind_data — This 32-bit field shall contain the object type of the referenced object, as specified in Table 5.9.

binding_type — This 8-bit field shall contain the type of binding, as specified in Table 5.9.

Table 5.9 Object Kind and Binding Type Codes

kind_data	binding_type	Description
0x64697200	0x02	DSM::Directory object
0x66696C00	0x01	DSM::File object
Other	Other	Reserved

IOP::IOR — This field shall contain an IOP::IOR, as specified in Table 5.2. If the IOR references a remote object, the referenced object shall not be a Service Gateway. It shall be either a directory or file in the remote TSFS directory structure. (The reason for this is that under the rules for binding names given in this section and Section 5.5.3, and the rules for associating URIs with objects given in Section 5.6, a link to a remote Service Gateway would result in URIs associated with the other objects in the remote TSFS that would not be well formed. See Section 5.6 for the definition of how URIs are formed from directory names, and see Annex B Section 5 for examples.)

childObjectInfo_length — This 16-bit field shall contain the length in bytes of the objectInfo structure for the object referenced by this directory entry.

ContentSize — This 64-bit field, which is only present when the referenced object is of type DSM::File, shall contain the number of bytes in the file content, or shall contain the value 0 to indicate that the number of bytes in the file content is not provided in this binding. (Note that even though ContentSize is a 64-bit field, the maximum size of a file in a TSFS is actually somewhat less than 256M bytes, since it must fit within a single data carousel module, which can consist of at most 64K blocks of slightly less than 4K bytes each.)

childObjectInfo_descriptor_byte — This 8-bit field shall contain a byte of the list of descriptors (i.e., the descriptor loop) that may appear in the objectInfo structure. See Section 6 for a definition of the descriptors that may appear here. A descriptor shall not be present in the objectInfo structure here unless it is also present in the objectInfo structure of the referenced object.

5.5.2 File Message Format

An object of type DSM::File shall be conveyed by a BIOP::FileMessage, as defined in Section 11.3.2.3 of [DSMCC] and further constrained by this standard. The BIOP::FileMessage is an instantiation of the Generic Object Message defined in [DSMCC]. The following rules constrain this instantiation:

- The objectKind field shall contain 0x66696C00 (the null-terminated ASCII string “fil”).
- The DSM::File::ContentSize attribute shall appear at the beginning of the fileObjectInfo field of the BIOP File message. Other attributes of the file conveyed in this message may be specified in the fileObjectInfo field via objectInfo descriptors defined in Section 6.
- The messageBody structure shall contain the BIOP::FileMessageBody structure, which is a stream of bytes containing the content of a file.

Table 5.10 defines the syntax of the DSM::FileMessage, as constrained by this standard.

Table 5.10 Syntax of File Message Format

Syntax	No. of Bits	Format
BIOP::FileMessage() {		
MessageHeader() {		
magic	4x8	uimsbf
biop_version.major	8	uimsbf
biop_version.minor	8	uimsbf
byte_order	8	uimsbf
message_type	8	uimsbf
message_size	32	uimsbf
}		
MessageSubHeader() {		
objectKey_length	8	uimsbf
for(j=0; j<objectKey_length; j++) {		
objectKey_byte	8	uimsbf
}		
fileobjectKind_length	32	uimsbf
fileobjectKind_data	32	uimsbf
fileObjectInfo {		
fileObjectInfo_length	16	uimsbf
ContentSize	64	uimsbf
for(j=0; j<fileObjectInfo_length - 8; j++) {		
fileObjectInfo_descriptor_byte	8	bslbf
}		
}		
serviceContextList_count	8	uimsbf
for (j=0; j<serviceContextList_count; j++) {		
serviceContext() {		
context_id	32	uimsbf
context_data_length	16	uimsbf
context_data_bytes	var	bslbf
}		
}		
}		
messageBody_length	32	uimsbf
FileMessageBody() {		
content_length	32	uimsbf
for(j=0; j< content_length; j++) {		
content_byte	8	bslbf
}		
}		
}		

- magic** — This field shall be set to 0x42494F50 to indicate that this is a BIOP message. The value of this field corresponds to the string “BIOP” encoded in ISO Latin-1.
- biop_version.major** — This 8-bit field shall be set to 0x01, indicating the major version number of the BIOP protocol used in this message.
- biop_version.minor** — This 8-bit field shall be set to 0x00, indicating the minor version number of the BIOP protocol used in this message.
- byte_order** — This 8-bit field shall be set to 0x00 (FALSE) indicating big-endian byte ordering for the subsequent fields of the message (including `message_size`).
- message_type** — This 8-bit field shall be set to 0x00, to indicate a BIOP object message. The values in the range from 0x01 to 0xFF are reserved by ISO/IEC 13818-6.
- message_size** — This 32-bit field shall contain the length in bytes of the `BIOP::FileMessage`, including all bytes following this field (not including this field) up to the end of the `BIOP::FileMessage`.
- ObjectKey_length** — This 8-bit field shall contain the length in bytes of the object key. The value shall be no greater than 4.
- ObjectKey_byte** — This 8-bit field shall contain a byte of the object key, which shall identify the object that is conveyed in this message uniquely within the data carousel module that contains this message. The value of the object key is only meaningful to the Broadcast Server and is treated as an opaque identifier (i.e., is not interpreted) by the Client.
- fileObjectKind_length** — This 32-bit field shall be set to 0x00000004, indicating that the `fileObjectKind_data` field is 4 bytes (32 bits) long.
- fileObjectKind_data** — This 32-bit field shall be set to 0x66696C00 (null terminated ASCII “fil”), indicating that this is a file object. It shall be identical to the Kind string present in all IORs referring to this file object.
- fileObjectInfo_length** — This 16-bit field shall specify the length in bytes of the `objectInfo` structure for this file object.
- ContentSize** — This 64-bit field shall contain the number of bytes in the file content, which is the same as the value of the `content_length` field below. (Note that even though this is a 64-bit field, the maximum size of a file in a TSFS is actually somewhat less than 256M bytes, since it must fit within a single data carousel module, which can consist of at most 64K blocks of slightly less than 4K bytes each.)
- fileObjectInfo_descriptor_byte** — This 8-bit field shall contain a byte of a list of descriptors (i.e., a descriptor loop). See Section 6 for the syntax and semantics of the descriptors that appear in this structure.
- serviceContextList_count** — This 8-bit field shall be set to the number of Service Context elements appearing in the Service Context List. Any Service Context elements have no meaning in this Standard.
- context_id** — This 32-bit field shall contain an identifier for the Service Context element.
- context_data_length** — This 16-bit field shall contain the length of the Service Context data.
- context_data_byte** — This 8-bit field shall contain a byte of the Service Context data.

messageBody_length — This 32-bit field shall contain the length of the FileMessageBody() structure following this field (not including this field). It shall be four (4) greater than the content_length field immediately following it.

content_length — This 32-bit field shall contain the length of the file following this field.

content_byte — This 8-bit field shall contain a byte of the file's data.

5.5.3 Service Gateway Message Format

The BIOP Service Gateway message is an instantiation of the generic object format. The following rules constrain this instantiation.

- The objectKind field shall contain the value 0x73726700 (the null-terminated ASCII string “srg”).
- The messageBody structure of ServiceGateway shall contain the BIOP::DirectoryMessageBody structure, defined in Section 5.5.1.
- The binding names in the directory shall conform to the syntax of an absolute URI, as specified in [URI], rather than the relative path syntax used in an ordinary directory object. Characters which are not present in US-ASCII, as well as reserved characters, shall be encoded by first taking the UTF-8 representation of these characters, and then escaping the octets of the UTF-8 representation using the %xx escape syntax defined by [URI] Section 2.1. For example, the relative URI component “café” would be encoded as “caf%c3%a9”, since the character “é” is encoded in UTF-8 as the octet sequence 0xC3 0xA9.

Thus, the BIOP Service Gateway message is syntactically and semantically identical to the BIOP Directory message, except that the objectKind field contains the string “srg” (followed by a null byte terminator), and the binding names in the message conform to the syntax of an absolute URI rather than the syntax of a relative path.

5.6 Uniform Resource Identifiers

For each directory path leading from the Service Gateway of a TSFS to an object of the TSFS, a URI shall be associated with the object, constructed by concatenating the binding names of the directory links in the path, omitting the terminating null character of each name and using a slash character (/) as a delimiter between the names. Thus, every object in the TSFS has one or more URIs associated with it, depending on the number of paths leading to it from the Service Gateway. Annex B Section 5 shows examples of this.

In keeping with the common semantics for URIs, whenever two objects in the same or different Transport Stream File Systems have the same URI associated with them, they shall be considered to be equivalent objects.

6. OBJECTINFO DESCRIPTORS

A binding is a data structure in a directory that binds a name to a referenced object. It often also includes some information about the object. In the case of a TSFS, a binding consists of a BIOP::Name structure, a binding_type, an IOP::IOR structure, and a childObjectInfo (or just objectInfo) structure.

In the objectInfo fields in an object message, and in the objectInfo fields of a binding structure, after any DSM-CC defined fields, one or more objectInfo descriptors may be carried to describe the attributes of the object.

6.1 Descriptor Identification and location

Table 6.1 lists the defined objectInfo descriptors and their allowable locations in various object messages. For each entry marked as “S”, the corresponding descriptor shall appear in the corresponding location. For each entry marked as “F”, the corresponding descriptor shall not appear in the corresponding location. For each entry marked as “O”, the corresponding descriptor may appear in the corresponding location, subject to the constraint that a descriptor may only appear in the objectInfo of a binding in a directory message when an identical copy of this descriptor is present in the objectInfo in the header of the object message for the object referenced by the binding as well.

Table 6.1 objectInfo Descriptors and Locations

Descriptor	Tag	Gateway	Directory	File	Binding	Description
Content Type	0x72	F	F	S	O	content (MIME) type
Time Stamp	0x81	O	O	S	O	last modified time
F = Forbidden, S = Mandatory, O = Optional						

Thus, both Content Type and Time Stamp descriptors shall appear in the fileObjectInfo of a File object message, a Content Type descriptor shall not appear in the dirObjectInfo of a Directory or Service Gateway object message, a Time Stamp descriptor may appear in the dirObjectInfo structure of a Directory or Service Gateway object message, and both Content Type and Time Stamp descriptors may appear in the objectInfo structure of a binding in a Directory or Service Gateway, but only if the same descriptors appear in the object message for the bound object.

6.2 Content Type Descriptor

The Content Type descriptor is defined in [Trigger] and reproduced in Table 6.2 for convenience. In the case of any discrepancy between the description in Table 6.2 and that appearing in [Trigger], the latter shall take precedence. This descriptor shall appear within the objectInfo of a DSM::File, and it may appear within the objectInfo of a binding referring to a file object. It shall not be present in the objectInfo within a DSM::Directory or DSM::ServiceGateway object message, or a binding referring to a DSM::Directory BIOP object.

Table 6.2 Syntax of ContentTypeDescriptor

Syntax	No. of Bits	Format
contentTypeDescriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (j=0; j<descriptor_length; j++) {		
content_type_data_byte	8	bslbf
}		
}		

descriptor_tag — This 8-bit field shall identify the descriptor. For the ContentTypeDescriptor it shall be set to 0x72.

descriptor_length — This 8-bit field shall specify the number of bytes in the identifier that follows.

content_type_data_byte — This 8-bit field shall contain a byte of a string that indicates the content (MIME) type of the object, as specified in [MIME]. (This is the string that would appear in the Content-Type field of a MIME message referring to an attachment of this content type.)

6.3 Time Stamp Descriptor

The syntax of a Time Stamp descriptor is defined in Table 6.3. The Time Stamp descriptor may be used to specify the time at which the object (to which the containing objectInfo structure belongs) was last modified. It shall appear in every DSM::File message, and it may appear in DSM::Directory and DSM::ServiceGateway messages. It may appear in a binding if it also appears in the object message to which the binding refers.

Table 6.3 Syntax of TimeStamp_descriptor

Syntax	No. of Bits	Format
TimeStamp_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
timeStamp	64	uimsbf
}		

descriptor_tag — This 8-bit field identifies this descriptor. For the TimeStamp_descriptor this field shall be set to 0xB9.

descriptor_length — This 8-bit field specifies the number of bytes of the descriptor that immediately follow the descriptor_length field (not including this field). The value of this field shall be set to 0x08.

timeStamp — This 64-bit unsigned integer quantity shall represent UTC time when the object was last modified, measured in milliseconds since 00:00:00 of January 1, 1970, GMT. The value 0xFFFFFFFFFFFFFFFF shall indicate that the time is not available. For a Directory or Service Gateway object, “modified” shall mean that a binding has been added or deleted, or a binding name has been changed. For a file, “modified” shall mean that there has been a change in the Content Type descriptor value, content_length, or one or more content_byte values. Note that the “epoch” or starting point for this time measurement is different from that used in the PSIP Event Information Tables. Note also that it is UTC time, not GPS time.]

7. TRANSPORT

Objects of a TSFS shall be transported in DSM-CC data carousels as defined in [DSMCC]. Each object (BIOP message) shall be transmitted in a single module of a DSM-CC data carousel. One module may contain one or more objects. Each module is fragmented into one or more blocks that are carried in DDB (DownloadDataBlock) messages as shown in Figure 7.1. Each DDB message is of the same size, except for the last block of the module, which may be of a smaller size. The Module delivery parameters, which are required to retrieve and assemble the objects, are sent in DII (DownloadInfoInitiate) messages. A DSI (DownloadServerInitiate) message provides the Carousel NSAP Address for the TSFS, and also provides an IOR that identifies the ServiceGateway object

for the TSFS. All of the DDB, DII, and DSI messages are encapsulated in DSM-CC sections, a special case of MPEG-2 sections, with `table_id` value 0x3C for DDB messages and `table_id` value 0x3B for DII and DSI messages, as specified in Section 9.2.2 of [DSMCC].

The DDB and DII messages that belong to a single data carousel, or download scenario, within an MPEG-2 program are distinguished by the fact that they all contain the same value in their `downloadId` field. (In the case of a DDB message the `downloadId` field is in the Download Data Header of the message. In the case of a DII message, the `downloadId` field is the first field of the message following the DSM-CC Message Header.) For a TSFS, the value of the `downloadId` field in these messages matches the value of the `carouselId` field in the `carouselNSAPAddress`.

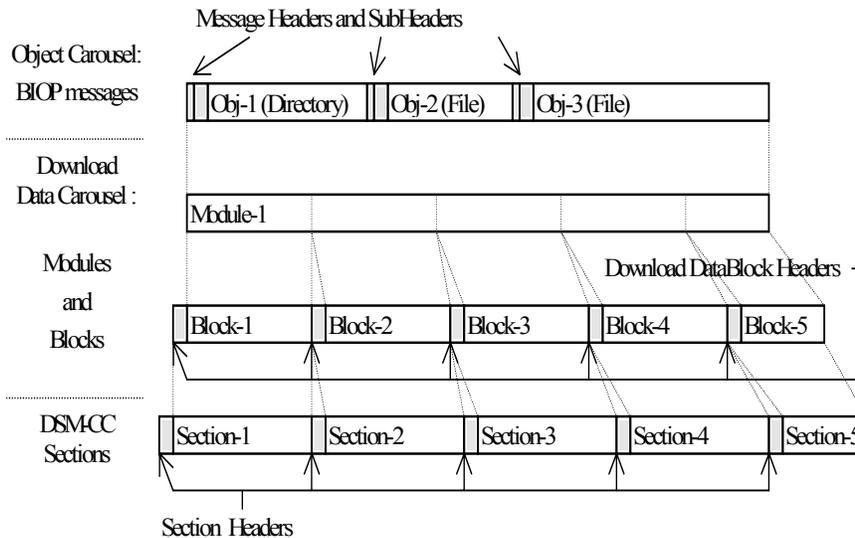


Figure 7.1 Encapsulation and fragmentation of BIOP messages.

7.1 DSM-CC Message Header

Both the DII messages and the DSI messages use the DSM-CC Message Header, which is defined in Section 2 of [DSMCC] and is reproduced in Section 7.2.3.1 of [A90]. The DSM-CC Message Header has a `transaction_id` field, which is used in [A90] to both identify and version the message for ATSC data carousels. In particular, `transaction_id[15..1]` (bits 1 through 15) serve as an identifier of the message, and `transaction_id[29..16]` (bits 16 through 29) serve as a version number of the message.

Any change in a module causes a change in the DII message that provides the delivery parameters for the module. This change in the DII message causes a change in its `transaction_id` value. Thus, by monitoring `transaction_id[15..1]` of a DII message, a receiver can detect whether any of the modules referenced by the DII message have changed.

This ATSC TSFS Standard adopts exactly the same usage of the `transactionId` field for DSM-CC data carousels containing a TSFS as that defined in [A90] for ATSC data carousels.

As described in Section 5.3, an IOR with a `BIOPProfileBody` references a DII message by means of a Tap in the `DSM::ConnBinder` component with a Message Selector giving the `transaction_id` of the DII message. Only the identification portion (bits 1 through 15) of the `transaction_id` field in the DII message is significant in identifying the correct DII message; i.e., a Tap in an IOR shall be

considered to reference a DII message if bits 1 through 15 of the transactionId field in the Message Selector of the Tap match bits 1 through 15 of the transactionId field of the DII message, regardless of whether the other bits of the transactionId match. The other bits of the transactionId may not match simply because the version of the DII message may have changed since the IOR was generated.

7.2 Transport of Service Gateway IOR

The IOR of the Service Gateway shall be broadcast by means of a DSI message. The syntax of the DSI message is defined by [DSMCC] and is shown in Table 7.1.

Table 7.1 Syntax of a DSI Message

Syntax	No. of Bits	Format
DownloadServerInitiate() {		
DsmccMessageHeader()		
ServerId	160	bslbf
CompatibilityDescriptor() {		
compatibilityDescriptorLength	16	uimsbf
}		
privateData_length	16	uimsbf
for (i=0; i<PrivateData_length; i++) {		
privateData_byte	8	uimsbf
}		
}		

ServerId — This 160-bit field shall contain the carouseINSAPaddress of this TSFS. Table 5.1 in Section 5.1 gives a detailed specification of the syntax and semantics of this field.

CompatibilityDescriptor — This 16-bit field shall be set to 0x0000, indicating that the compatibility descriptor is not used in this message.

privateData_length — This 16-bit field shall specify the number of bytes in the PrivateData_bytes field.

privateData_byte — This 8-bit field shall contain a byte of the BIOP ServiceGatewayInfo structure, as specified by [DSMCC] and described for convenience in Table 7.2.

Table 7.2 Syntax of BIOP: ServiceGatewayInfo

Syntax	No. of Bits	Format
ServiceGatewayInfo() {		
IOR::IOR()	variable	bslbf
downloadTaps_count	8	uimsbf
for(k=0; k<downloadTaps_count; k++) {		
DSM::Tap() {		
tapId	16	uimsbf
tapUse	16	uimsbf
associationTag	16	uimsbf
selector()	variable	bslbf
}		
}		
serviceContextList_count	8	uimsbf
for (j=0; j<serviceContextList_count; j++) {		
serviceContext() {		
context_id	32	uimsbf
context_data_length	16	uimsbf
for (i=0; i<context_data_length; i++) {		
context_data_byte	8	bslbf
}		
}		
}		
userInfo_length	16	uimsbf
for (i=0; i<userInfo_length; i++) {		
userInfo_byte	8	bslbf
}		
}		

IOR::IOR — This field shall contain the IOR of the Service Gateway object for this TSFS. It shall contain a BIOPProfileBody, rather than a LiteOptionsProfileBody. See Table 5.2 for a specification of the syntax and semantics for an IOR containing a BIOPProfileBody.

downloadTaps_count — This 8-bit field shall specify the number of Taps included in this ServiceGatewayInfo structure. One or more Taps may be included, each referencing DII messages and/or DDB messages for the modules containing objects of the TSFS, to facilitate acquisition of the entire TSFS. It is not required that any Taps be included.

tapId — The value of this 16-bit field shall be set to 0xFFFF, indicating that the usage of this field is reserved.

tapUse — As specified in Sections 11.2.6 and 11.3.3.3 of [DSMCC], this 16-bit field shall be set to 0x0007 (DOWNLOAD_CTRL_DOWN_USE) for any Tap in the list that references a DII message giving delivery parameters for modules containing objects of the TSFS, and it shall be set to

0x000A (DOWNLOAD_DATA_DOWN_USE) for any Tap in the list that references a program element containing DDB messages delivering modules containing objects of the TSFS. Taps with any other value of tapUse have no meaning in this standard.

associationTag — This 16-bit field identifies a Program Element listed in the Program Map Section for the current virtual channel. The value of this field shall be identical to the association_tag value of an association_tag_descriptor in the Program Map Section for the current virtual channel.

selector() — The selector() structure is defined in Section 5.6.1 of [DSMCC] and shown in Table 12.8 of [A90]. When the value of tapUse is 0x0016 (BIOP_DELIVERY_PARA_USE), the selector() shall be a MessageSelector, as defined in [DSMCC] and described in Table 5.5. This selector() specifies which of the possibly multiple DII messages in the Program Element identified by the Tap is being referenced by the Tap. When the value of tapUse is 0x0017 (BIOP_OBJECT_USE), the selector() shall consist solely of an 8-bit selector_length field, which shall be set to 0x00 to indicate that the remainder of the selector() is empty.

serviceContextList_count — This 8-bit field shall be set to the number of Service Context elements appearing in the Service Context List. Any Service Context elements have no meaning in this standard.

context_id — This 32-bit field shall contain an identifier for the Service Context element.

context_data_length — This 16-bit field shall contain the length of the Service Context data.

context_data_byte — This 8-bit field shall contain a byte of the Service Context data.

userInfo_length — This 16-bit field shall specify the number of bytes carried in the user information data field.

userInfo_byte — This 8-bit field shall contain a byte of the user information data carried by this service gateway object. If present, the user information data have no meaning in this standard.

7.3 Transport of Module Delivery Parameters

The Module delivery parameters for each Module carrying objects of the TSFS shall be encapsulated in DII messages. The syntax of the DII message is defined by [DSMCC] and is shown in Table 7.3.

Table 7.3 Syntax of a DII Message

Syntax	No. of Bits	Format
DownloadInfoIndication () {		
DsmccMessageHeader()		
downloadId	32	uimsbf
blockSize	16	uimsbf
windowSize	8	uimsbf
ackPeriod	8	uimsbf
TCDownloadWindow	32	uimsbf
TCDownloadScenario	32	uimsbf
CompatibilityDescriptor() {		

compatibilityDescriptorLength	16	uimsbf
}		
numberOfModules	16	uimsbf
For(l=0; l<numberOfModules; l++) {		
moduleId	16	uimsbf
moduleSize	32	uimsbf
moduleVersion	8	uimsbf
ModuleInfo_length	8	uimsbf
ModuleInfo {		
moduleTimeOut	32	uimsbf
blockTimeOut	32	uimsbf
minBlockTime	32	uimsbf
numberTaps	8	uimsbf
for (j=0; j<numberTaps; j++) {		
DSM::Tap() {		
tapId	16	uimsbf
tapUse	16	uimsbf
associationTag	16	uimsbf
selector()	variable	bslbf
}		
}		
userInfo_length	8	uimsbf
for (j=0; j<userInfo_length; j++) {		
userInfo_byte	8	bslbf
}		
}		
}		
privateData_length	16	uimsbf
for (i=0; i<privateData_length; i++) {		
privateData_byte	8	bslbf
}		
}		

DsmccMessageHeader() — This structure shall be the message header as specified in [DSMCC].

downloadId — This 32-bit field shall be set to the carouselId of this TSFS. All objects of a single TSFS shall be carried in modules of the same download scenario. All DownloadDataBlock and DownloadInfoIndication messages in the download scenario share the same value of downloadId.

blockSize — This 16-bit field shall be set to the length in bytes of the data in the blocks in each module carried in the DownloadDataBlock messages referenced by this DII message, as specified in [DSMCC]. Every block in each module except the last one shall have this same size. The last block may be smaller.

- windowSize** — This 8-bit field shall be set to 0x00, as specified by [DSMCC].
- ackPeriod** — This 8-bit field shall be set to 0x00, as specified by [DSMCC].
- TCDownloadWindow** — This 8-bit field shall be set to 0x00, as specified by [DSMCC].
- TCDownloadScenario** — This 32-bit field shall be set to 0x000000, as specified by [DSMCC], to indicate that this field is not used for object carousels.
- compatibilityDescriptorLength** — This 16-bit field shall be set to 0x00, indicating that the compatibility descriptor is not used in this message.
- numberOfModules** — This 16-bit field shall contain the number of modules referenced by this DII message, as specified in [DSMCC].
- moduleId** — This 16-bit field shall contain the identifier for this module, as specified in [DSMCC]. The moduleId shall be unique within the download scenario (i.e., within the set of modules in the same virtual channel with the same value of downloadId).
- moduleSize** — This 32-bit field shall contain the total size in bytes of this module, as specified in [DSMCC].
- moduleVersion** — This 8-bit field shall contain the version number of this module, as specified in [DSMCC].
- moduleInfo_length** — This 8-bit field shall specify the number of bytes in the moduleInfo structure, as specified in [DSMCC].
- moduleTimeOut** — This 32-bit field shall contain the timeout value in microseconds that may be used to time out the acquisition of all blocks of the module. The value 0 shall mean that no timeout value is specified.
- blockTimeOut** — This 32-bit field shall contain the timeout value in microseconds that may be used to time out the next block of the module after a block has been acquired. The value 0 shall mean that no timeout value is specified.
- minBlockTime** — This 32-bit field shall contain the minimum time period that exists between the delivery of two subsequent blocks of the module. The value 0 shall mean that no minimum time period is specified.
- numberTaps** — This 8-bit field shall contain the number of Taps in the ModuleInfo structure. Any Taps beyond the first one have no meaning in this standard.
- tapId** — The value of this 16-bit field shall be set to 0xFFFF, indicating that the usage of this field is reserved.
- tapUse** — For the first Tap in the ModuleInfo structure, this 16-bit field shall be set to 0x0017 (BIOP_OBJECT_USE).
- associationTag** — The value of this field shall be identical to the association_tag value of an association_tag_descriptor in a Program Map Section for the virtual channel containing this DII message. For the first Tap in the ModuleInfo structure, the program element associated with this descriptor shall be the program element carrying this module.
- selector()** — For the first Tap in the ModuleInfo structure, the selector() shall consist solely of an 8-bit selector_length field, which shall be set to 0x00 to indicate that the remainder of the selector() is empty.
- userInfo_length** — This 8-bit field shall specify the number of user information bytes immediately following this field (not including this field).

userinfo_byte — This 8-bit field shall contain one byte of the user information bytes in the ModuleInfo structure. Any user information bytes present have no meaning in this standard.

privateData_length — This 16-bit field shall specify the number of private data bytes following this field.

privateData_byte — This 8-bit field shall contain one byte of the private data carried by this DII message. Any private data present have no meaning in this standard.

7.4 Semantics of TransactionId

This section defines the semantics of the transactionId field in the DSM-CC Message Header for DSI and DII messages of a TSFS, which are the same as the semantics specified in [A90] for ATSC data carousels.

The transactionId has a dual role, providing both identification and versioning mechanisms for control messages; i.e. DownloadInfoIndication and DownloadServerInitiate messages. According to the DSM-CC specification, the transactionId must uniquely identify a download control message within a data carousel; however it must also be “incremented” whenever any field of the message is modified. The term “incremented”, originally defined in the DSM-CC specification, should be interpreted as “changed” within this specification.

The object carousel is carried on top of a data carousel, which in this standard means a DSI message, a set of DII messages, and a set of DDB messages carrying the modules described in the DII messages. The DII and DDB messages all have a common downloadId value. The DDB messages may be carried in a different program element from the DSI and DII messages.

When a module is changed, the version number of the module needs to be changed. This implies that the DownloadInfoIndication message that references the module needs to be also updated, since it contains the module version number. Since the DownloadInfoIndication is updated, its transactionId needs to be also changed. However, it would be highly undesirable to be forced on account of this to change all the IORs in Directory bindings that reference the DII messages by means of their transactionIds, since this would cause additional modules to be updated, which in turn would require the transactionIds of additional DII messages to be updated, and so on. It is desirable to avoid the need to update additional messages, and to limit the implications of updating a module to the module itself and the DownloadInfoIndication message referencing the module.

To address the above issues, the transactionId shall consist of four sub-fields, as specified in [A90] and described in Table 7.4. This reflects the dual role of the transactionId (outlined above) and the constraints imposed to reduce the effects of updating a module.

Table 7.4 TransactionId Sub-Fields

Subfield	Bits	Description
updateFlag	0	This field shall be toggled each time a control message is updated.
identification	1-15	For DSI messages, this field shall be set to 0x0000. For DII messages, it shall be set to a non-zero value.
version	16-29	This field shall be changed each time a control message is updated.
originator	30-31	This field shall be set to '10', indicating that the transactionId is set by the server.

The identification subfield shall be unique within the scope of all DII messages in the data carousel. When the message changes, the `updateFlag` shall be toggled, and the version subfield shall be changed (typically incremented modulo the size of the subfield, but this is not required).

Any change to a Module results in incrementing its `moduleVersion` field. This change is reflected in the corresponding field in the description of the Module in the `DownloadInfoIndication` message that gives its delivery parameters. Since a field in the `DownloadInfoIndication` message is changed, the `updateFlag` and version subfields of its `transactionId` field are changed to indicate a new version of the message. If the entire `transactionId` field were used to identify the message, then this would require that every message referencing the `transactionId` would also have to be updated. However, this massive propagation of updates is not needed, since the identification subfield of the `transactionId` field is not affected by these updates.

When locating a message based on the `transactionId` value used for referencing the message, only the identification part (bits 1..15) needs to be matched. Thus, the referencing messages need not be updated every time a referenced control message is updated. The implications of updating a module are limited to the module itself and the `DownloadInfoIndication` message describing the module.

A receiver can find out if a particular module that it has retrieved earlier has changed, by filtering the `DownloadInfoIndication` message that described that module and checking if it has been changed.

7.5 Signaling of Transport Stream File Systems

A Transport Stream File System used by a data service application shall be signaled in the Data Service Table (DST) of the data service by a Tap with associated parameters and structure as described in Table 7.5. This Tap signals the location of the DSI message that carries information about the Service Gateway of the TSFS, and optionally uses a URI in the selector structure to restrict the referenced resource to a specified file or directory of the TSFS. The individual program elements carrying all the DII and DDB messages of the TSFS shall not be signaled in the DST. (No mechanism is provided in this standard to signal these program elements in the DST. Note, however, that they may all be signaled in the DSI message for the TSFS.)

The `protocol_encapsulation` value 0x0F shall be used in the Data Service Table (DST) to signal the TSFS (Object Carousel) encapsulation, extending Table 12.5 of [A90].

In the event that the data service uses a remote TSFS, either directly or by indirect reference via a remote object reference from a local TSFS, the remote program element carrying the DSI message for the remote TSFS shall be signaled in the Network Resources Table (NRT), and a Tap in the DST shall reference this entry in the NRT. The same protocol encapsulation value shall be used for such Taps.

The following Table is an excerpt of the Data Service Table defined in [A90], showing how a Tap and its related fields need to be set up when `protocol_encapsulation` is equal to the value signaling a TSFS.

Table 7.5 DST Tap Referencing a Transport Stream File System

Syntax	No. of Bits	Format
Tap_with_related_fields() {		
protocol_encapsulation	8	uimsbf
action_type	7	uimsbf
resource_location	1	bslbf
Tap() {		
tap_id	16	uimsbf
use	16	uimsbf
association_tag	16	uimsbf
selector()	variable	bslbf
}		
tap_info_length	16	uimsbf
for(k=0; k<N; k++) {		
descriptor()	variable	bslbf
}		

protocol_encapsulation — If the Tap() references a Transport Stream File System, this 8-bit field shall have value 0x0F.

action_type — This 7-bit field shall be set as specified in [A90].

resource_location — This 1-bit field shall be set as specified in [A90].

tap_id — This 16-bit field is used to identify the resource, as specified in [A90].

use — This 16-bit field shall have value 0x0000, as specified in [A90].

association_tag — This 16-bit field shall uniquely specify either a program element in a Program Map Table section for the MPEG-2 program containing the data service, or a DSM-CC Resource Descriptor listed in the Network Resource Table, as specified in [A90]. If the Tap() references a Transport Stream File System, the specified program element shall contain the DownloadServerInitiate (DSI) message that points to the ServiceGateway for the referenced Transport Stream File System.

selector() — If the Tap() references a TSFS, the selector() structure shall be a TSFS_selector(), as described in Table 7.6.

tap_info_length — This 16-bit field shall be set as specified in [A90].

descriptor() — If present, this structure shall be a descriptor, as specified in [A90].

For a Tap() that references a TSFS, this ATSC TSFS Standard defines a selector structure of a selector_type 0x109, extending Table 12.9 of [A90]. This selector structure provides the carouselId of the TSFS (which is the carouselId in the carouselNSAPAddress that appears in the DSI message pointing to the Service Gateway of the TSFS), the timeout interval for acquiring the DSI message, and optionally a URI that specifies a file or directory of the TSFS (with the semantics that the Tap() reference is limited to the specified file or directory). The specification of this selector() appears in Table 7.6.

Table 7.6 TSFS Selector for Tap Referencing a TSFS

Syntax	No. of Bits	Format
TSFS_selector() {		
selector_length	8	uimsbf
selector_type	16	uimsbf
carouselId	32	uimsbf
DSI_timeout	32	uimsbf
if(selector_length > 10) {		
for (k = 0; k<selector_length-10; k++) {		
URI_byte	8	bslbf
}		
}		
}		

selector_length — This 8-bit field shall specify the number of bytes in the selector following this field (not including this field). Its value shall be 10 when the selector specifies only a carouselId and a DSI timeout interval, and it shall be $10 + N$ when the selector specifies a carouselId, a DSI timeout interval, and a URI, where N is the number of bytes in the string representing the URI.

selector_type — This field shall have value 0x109, indicating that the selector is a TSFS_selector().

carouselId — This 32-bit field shall specify the carouselId of the DownloadServerInitiate message that contains the IOR of the ServiceGateway for the referenced Transport Stream File System.

DSI_timeout — This 32-bit field shall specify the timeout interval in microseconds for the DownloadServerInitiate message that contains the IOR of the BIOP ServiceGateway object for the referenced Transport Stream File System, or shall have value 0 to indicate that no timeout interval is specified.

URI_byte — If selector_length is greater than 10, then this field shall contain a byte of a string representing an absolute URI, as specified in [URI]. The URI shall be identical to the URI associated with a file or a directory of the Transport Stream File System, as defined in Section 5.6 of the ATSC TSFS Standard, with the semantics that only that file or the content of that directory is referenced by this Tap. A process for resolving this URI to locate the specified file or directory is described in Annex B Section 5.

Any program element containing any messages of a TSFS shall be labeled with stream_type 0x0B in the MPEG-2 Program Map Section and in the Service Location Descriptor of the Virtual Channel Table.

7.6 Private Usage Collision Avoidance

If a Tap in a DST signals a TSFS in which any user private data structures appear (i.e., data structures that are not defined by this standard, such as userInfo in the ServiceGatewayInfo structure of the DSI message, or privateData in a DII message, or userInfo associated with a module in a DII message), then an MPEG-2 defined Registration Descriptor (MRD) shall appear in the descriptor loop associated with that Tap [MRD]. The MPEG-2 Registration Descriptor contains a 32 bit format identifier field (administered by SMPTE) which is registered to the organization defining

the syntax of the user private data structures. The proper registration of a format identifier value allows unique identification of the semantic meaning of the contents of the private fields to allow collision avoidance. Only one MRD shall be allowed in a given Tap loop [COLLISION]. A DST conveying an MRD in the Tap loop must be received before parsing of the user private fields can begin.

Annex A: Carousel Design (Informative)

A DSM-CC Data Module may contain one or more TSFS objects, of any type (hence the need for ObjectKey). The objects may be organized into the modules in many different ways, depending on application needs.

For maximum efficiency of data access in certain types of applications, designers of a TSFS should only compile two types of module:

- **Directory module:** contains at most one DSM::ServiceGateway object and one or more DSM::Directory objects (i.e., cannot contain DSM::File objects).
- **File module:** contains one or more DSM::File objects (i.e., cannot contain any DSM::ServiceGateway or DSM::Directory objects).

I.e., the BIOP service gateway and BIOP directory objects should be encapsulated in one or several data carousel modules that are separate from the data modules conveying the BIOP file objects. The data modules conveying the BIOP directory objects may be transmitted more often to allow receivers to reconstruct the file hierarchy with minimum latency. To reduce latency induced by parsing a data module payload without impacting coding efficiency too much, it is possible to map any large BIOP file object to a single data module.

Efficiency may be further improved by placing all files belonging to a single application in a single directory and in a single file module.

For other types of applications, it may be more efficient to have each module contain an entire URI name space (i.e., all the objects with a common base-URI), consisting of a directory one level below the Service Gateway together with all of its subdirectories and files, so that only a single module needs to be acquired in order to access the entire URI name space. Many other organizational structures are possible.

Annex B: Object and File System Acquisition (Informative)

1. INTRODUCTION

An IOR (Interoperable Object Reference) is a structure containing necessary information for locating a TSFS object. Two types of IORs are specified in this TSFS standard, one containing a `BIOPProfileBody` that is used to reference objects in the same TSFS, and one containing a `LiteOptionsProfileBody` that is used to reference objects in a remote TSFS (which may be in the same virtual channel, a different virtual channel in the same transport stream, or a different virtual channel in a different transport stream).

This annex describes how to acquire a local TSFS object given its IOR, how to acquire a remote TSFS object given its IOR, how to acquire an entire TSFS given its Tap in the Data Service Table, and finally how to acquire any object given its URI. Several of the figures in this Annex were adapted from figures in [DVB-DB].

2. LOCAL OBJECT ACQUISITION

An IOR that refers to an object within the same TSFS (Object Carousel) contains a `BIOP Profile Body` (with syntax as shown in Table 5.3) that contains the information necessary to locate the `BIOP` messages that convey the object data and attributes. This `BIOP Profile Body` comprises an `ObjectLocation` component and a `ConnBinder` component.

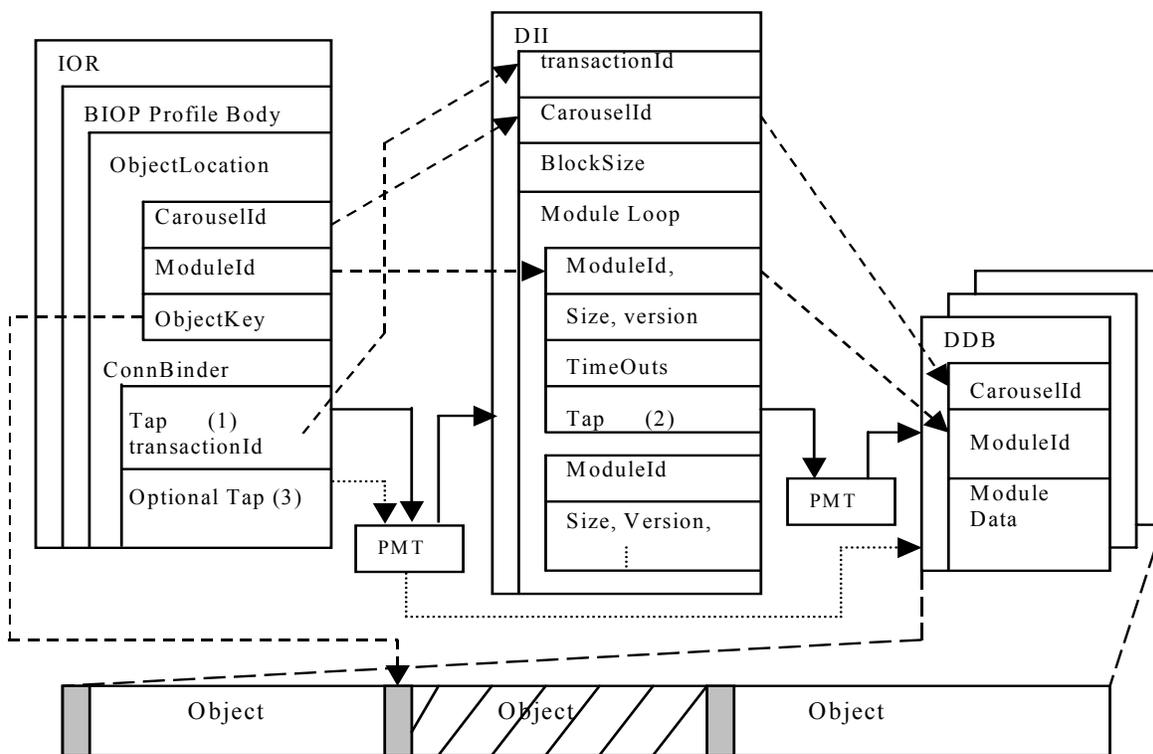
The `ConnBinder` component always includes one Tap that tells where to find the `DII` message that conveys the module delivery parameters of the module containing the object. The `tapUse` field of this Tap is set to `BIOP_DELIVERY_PARA_USE` (0x0016). The `tapId` field of this tap is not used. The `associationTag` field of this Tap is an indirect reference to the program element containing the `DII` message. The `selector` field of the Tap gives the `transactionId` of the `DII` message and also the timeout interval for acquiring the `DII` message; i.e., the maximum interval between transmissions of the message.

The `ConnBinder` component may also include a Tap that tells where to find the `DDB` messages that convey the module containing the object. The `tapUse` field of this Tap is set to the value `BIOP_OBJECT_USE` (0x0017). The `tapId` field is not used. The `associationTag` is an indirect reference to the program element containing the `DDB` messages. The `selector` field of the Tap is empty. This information is also contained in the `DII` message, but sometimes it is possible to use direct information about the `DDB` messages in the `ConnBinder` to shortcut the acquisition process by not waiting to acquire the `DII` message.

The reason for using an `associationTag` to identify a program element, rather than just giving its `PID`, is that a `PID` may change during remultiplexing of Transport Streams, but an `associationTag` remains associated with the same program element. The way this works is that an `association_tag_descriptor` is attached to the program element's entry in the Program Map Table (PMT) section for the virtual channel, and the `associationTag` value in the Tap is to be matched with the `associationTag` value in the descriptor to select the program element. Any remultiplexing operation leaves this descriptor attached to the same program element entry in the PMT, so the correct program element will be selected even after remultiplexing.

The ObjectLocation component contains the three values carouselId, moduleId, and objectKey, where the carouselId is identical to the downloadId value that appears in the both the DII and DDB messages, the moduleId is a unique identifier of the module within the carousel, and the objectKey is a unique identifier of the object within the module. Figure B1 illustrates the process of using the information in the IOR to acquire the object.

A DII message is uniquely determined within a virtual channel by its downloadId and transaction_id. Thus, the receiver can use the associationTag value in the BIOP_DELIVERY_PARA_USE Tap of the ConnBinder component to identify the program element containing the DII message, and then use the transaction_id in this Tap and the carouselId from the ObjectLocation component to identify the DII message within the program element. It can then use the moduleId to identify the entry in the DII message giving the delivery parameters for the DDB messages carrying the desired module.



- 1) TapUse = BIOP_DELIVERY_PARA_USE, selector = {transactionId, timeout}, associationTag points via PMT to program element carrying DII message.
- 2) TapUse = BIOP_OBJECT_USE, selector = none, associationTag points via PMT to program element carrying DDB messages.
- 3) Same as the Tap usage of (2).

Figure B1 Resolving an object from its IOR with BIOP profile body.

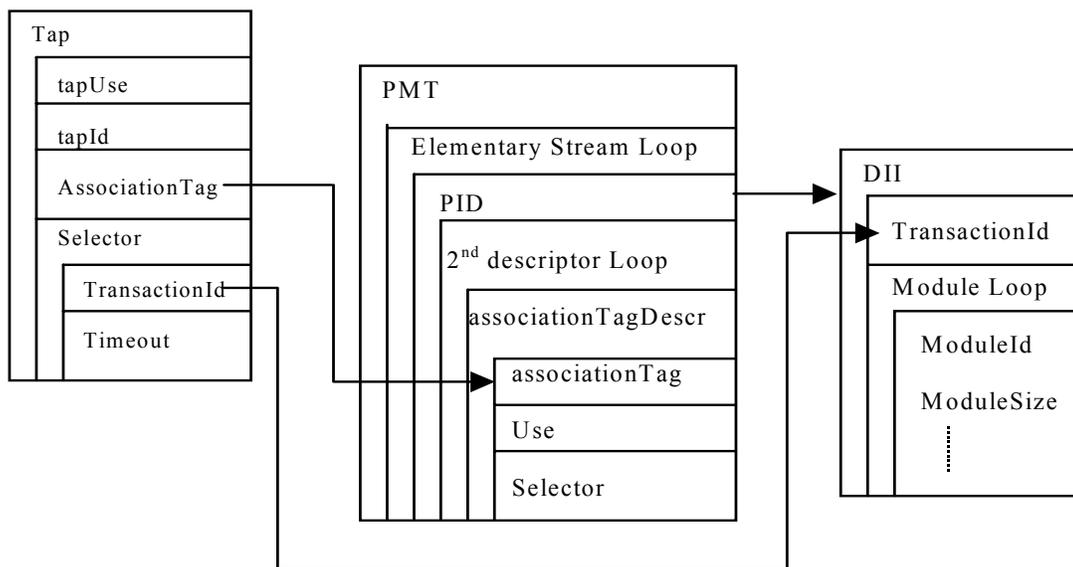
From this entry in the DII message the receiver can get a Tap identifying the program element containing the DDB messages, and it can use the carouselId and moduleId to identify these DDB messages within that program element. It also gets the block size (size of the payload of

each DDB message), total module size, module version, and various timeout parameters. With this information it can allocate buffers of appropriate size, make sure it has acquired all the blocks of the module, and use the timeout intervals to detect errors in transmission.

Once the receiver has acquired the module containing the object, it can proceed to parse the module from the top down looking for the correct object, identified by its objectKey. Efficient parsing is implemented by starting at the top and inspecting the first 13 + *N* bytes of each BIOP Object Message (*N* is the number of bytes taken by each objectKey field). If the objectKey is not the desired value, the value of the message_size field in the Object Message is used to get the start index of the next BIOP object message in the same data module. This is done until an objectKey in the MessageSubHeader structure of a target BIOP Object Message is found that matches the desired value.

A receiver may opt to follow a more aggressive acquisition of the data module when the IOR of the object includes a ConnBinder component with tapUse value of BIOP_OBJECT_USE that allows direct acquisition of the object. However, there are some pitfalls to this approach. In this situation the receiver does not know the block size, any of the timeout values for the DDB messages, or the total module size. Thus, it may have trouble allocating buffers appropriately, it may end up waiting indefinitely in the event of transmission errors, or giving up prematurely while waiting for the module, and it may not be able to determine conclusively when it has the complete module. If it encounters a short block, it knows that must be the last block of the module, but if by chance the last block is the same size as the others, the receiver cannot tell if it has them all or not.

Figure B2 gives a more detailed view of the process of using the PMT to identify the program element referenced by a Tap.



tapUse = BIOP_DELIVERY_PARA_USE.

Figure B2 Use of Tap for locating a program element.

3. REMOTE OBJECT RESOLUTION

An IOR referencing an object from another broadcast Service Domain (another TSFS) contains a Lite Options Profile Body (with syntax as shown in Table 5.4) that provides the information necessary to locate the Service Gateway of the remote TSFS, plus the logical directory path in that TSFS from the Service Gateway to the object itself. This information comes in the form of a ServiceLocation component that provides a carouselNSAPaddress and a sequence of bindings representing a path from the Service Gateway to the referenced object.

The NSAP address is a unique identifier for the Service Domain. This identifier includes TransportStreamID, originalTSID, program_number, source_id, originalSourceId, and carouselId fields that allow a receiver to locate the Service Gateway for the Service Domain. The originalTSID and originalSourceId fields are the transportStreamID and source_id of the virtual channel in which the remote TSFS was originally broadcast. The transportStreamID and source_id fields are the current Transport Stream ID and source_id, to the extent that these are known to the server that generated the IOR. (The resolution process would be much more straightforward if there was no possibility that the remote transport stream may have been involved in a remultiplexing operation without the knowledge of the server that generated the IOR.) The carouselId field gives the carouselId of the TSFS, which is scoped by the virtual channel and therefore should not be affected by remultiplexing operations.

As illustrated in Figure B3, a receiver goes through the following steps to locate a Service Gateway from its NSAP address in the case when the transportStreamID and program_number in the NSAP address are correct, either because there has been no remultiplexing of the target virtual channel, or because the NSAP address has been corrected in some way to reflect the remultiplexing:

- 1) In the receiver's cached channel map locate the transport stream corresponding to the transportStreamID in the NSAP address, tune to that transport stream, and acquire its PAT.
- 2) In the program loop of that PAT locate the entry for the MPEG-2 program corresponding to the program_number in the NSAP address.
- 3) Using the PMT_PID field of that entry, acquire the PMT section for that program.
- 4) Upon acquiring that PMT section, double check that the program_number in that PMT section matches the program_number being looked for.
- 5) Find the PID of the program element containing the DST by looking in the stream loop of the PMT section for a program element with stream_type 0x95, and acquire the DST.
- 6) Find the Tap in the DST with encapsulation_protocol value 0x0F and carouselId value in its selector field matching the carouselId value in the NSAP address.
- 7) Look at the association_tag value of this Tap and locate the stream entry in the PMT section with an association_tag descriptor that has a matching association_tag value.
- 8) The PID for this entry identifies the program element that contains the DSI message for the desired Service Domain.
- 9) Acquire from this program element the DSI message that has a carouselId matching the carouselId of the NSAP address.
- 10) Use the Service Gateway IOR in this DSI message to acquire the Service Gateway object, as described in Annex B.

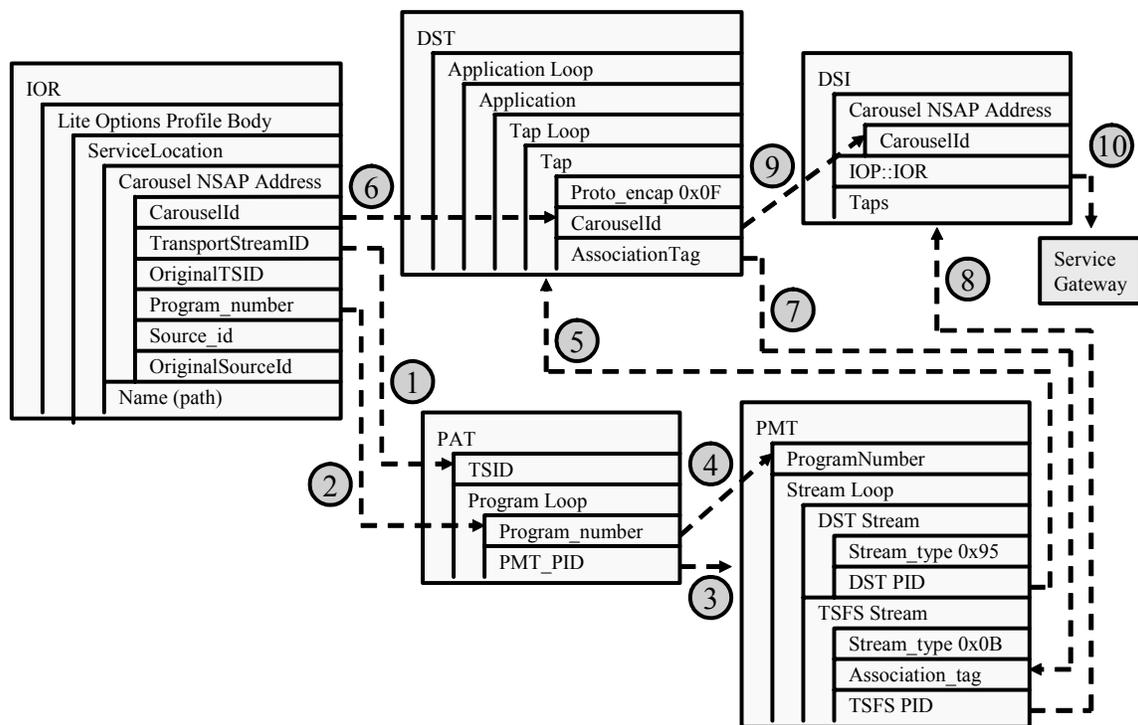


Figure B3 Finding service gateway from NSAP address (no remultiplexing case).

Once the Service Gateway is acquired, it is a simple matter to walk down the directory hierarchy to the desired object, matching the next name in the desired path to a directory entry at each step, picking up the corresponding IOR, and resolving it according to the process in Section 2 of this annex to get to the next object down the hierarchy.

If the “path” was in the form of a single URI, instead of a sequence of path terms, the process is slightly more complex. This process is described in Section 5 of this annex.

The resolution process becomes more complex if the value of `transportStreamID` and possibly `source_id` of the remote TSFS have changed because of a remultiplexing operation, and the server that generated the IOR containing the NSAP Address was not aware of the change. In such a situation the receiver will not be able to find a match for the `transportStreamID` in the channel map, since transport streams should be unique at a regional level, and a remultiplexed transport stream should never retain the transport stream of any of the original input streams.

If the `source_id` is in the range 0x1000 through 0xFFFF, then it is guaranteed to be unique at the regional level, as specified in [PSIP], and there is no real problem. The receiver can match the `source_id` in the carousel NSAP address with a `source_id` in its channel map, taken from VCT tables, and this should give the correct virtual channel, regardless of the value of transport stream

ID. This then gives the receiver the correct Transport Stream ID and correct MPEG-2 program number. Figure B4 illustrates the process for acquiring the Service Gateway in this case.

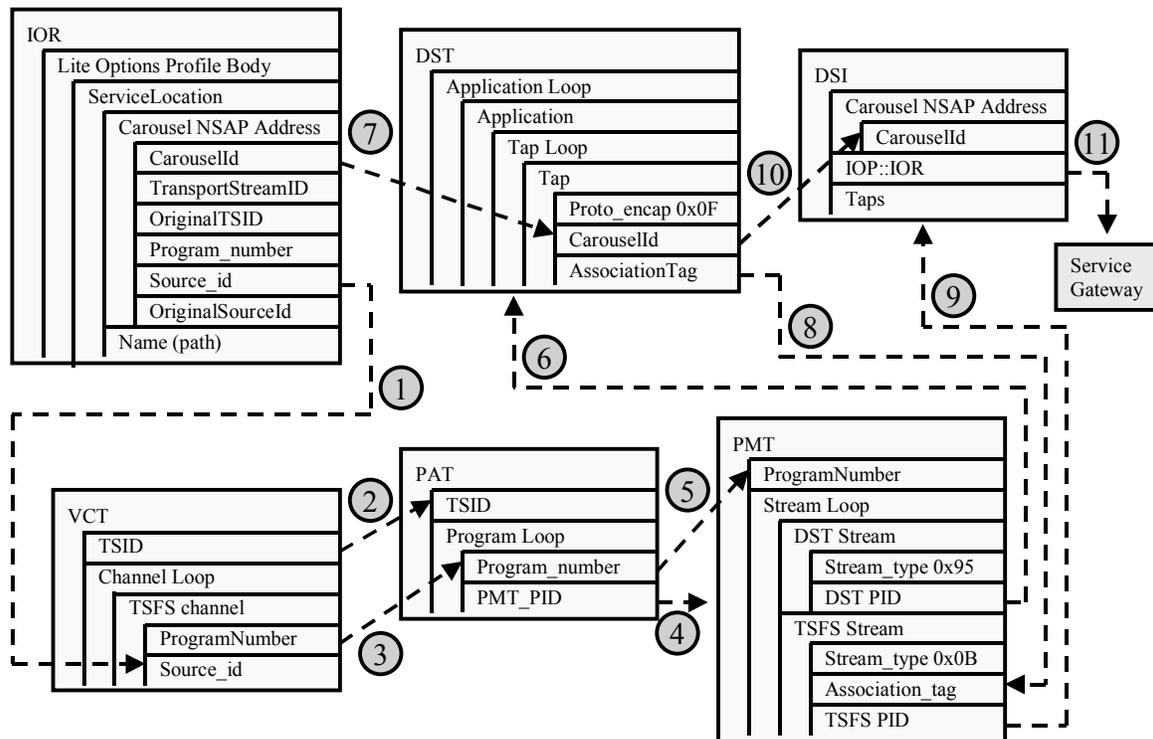


Figure B4 Finding service gateway from NSAP address (remultiplexed case).

If the `source_id` is not unique at the regional level, then the receiver really has no choice but to scan in real time through all the virtual channels it can receive, acquire the Data Service Tables for all of them that have a data service, look for Taps referencing a TSFS, check for a match of the `carouselId` in such Taps against the `carouselId` in the carousel NSAP address, and acquire the DSI message for any that match. If the `originalTSID` and `originalSourceId` in the `serverId` field of the DSI message match those of the carousel NSAP address of the IOR being resolved, then the correct TSFS has been found, and the directory walking process can continue as before.

4. TRANSPORT STREAM FILE SYSTEM ACQUISITION

Figure B5 illustrates the sequence for acquiring all the objects in a TSFS with a simple hierarchical directory structure. The IOR of the Service Gateway object can be extracted from the DSI message. Given the IOR for the Service Gateway, the Service Gateway object can be parsed from the data module carrying it, as in Section 2 of this annex. From the binding structure inside the Service Gateway object, the IORs for Directory object D_0 and File object F_2 can be

extracted. Given the IOR for F_2 , the file data for F_2 can be acquired from the data module carrying the F_2 object, as in Section 2 of this annex. Given the IOR for D_0 , Directory object D_0 can be extracted, from which the IORs for F_0 and F_1 can be obtained. Similarly, given the IOR for F_0 and the IOR for F_1 , the file data for F_0 and F_1 can be acquired. Thus, the IORs for objects in the directory structure are acquired in the following sequence: IOR-SG, IOR- D_0 , IOR- F_2 , IOR- F_0 and IOR- F_1 . Once an IOR of an object is obtained, the object can then be retrieved from the stream.

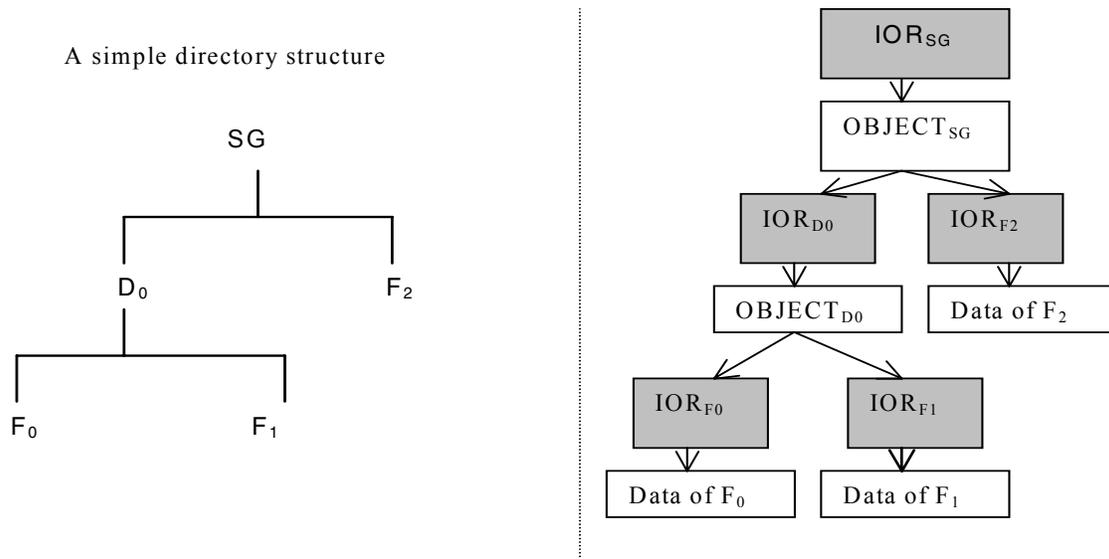


Figure B5 How to acquire objects in a simple directory structure.

If the directory structure is not a strict tree, but rather a more general acyclic graph, the process is just a little more complex. Some bookkeeping is required to avoid acquiring the same object more than once, and to keep track of the multiple URIs associated with some objects (resulting from the multiple paths leading to them from the Service Gateway).

5. URI RESOLUTION

Figure B6 illustrates a TSFS directory structure involving two Transport Stream File Systems (two Service Gateways), with a number of local directory links within each one and one remote directory link between them. The binding name is shown next to each directory link, and the resulting URIs associated with the files are shown below the files.

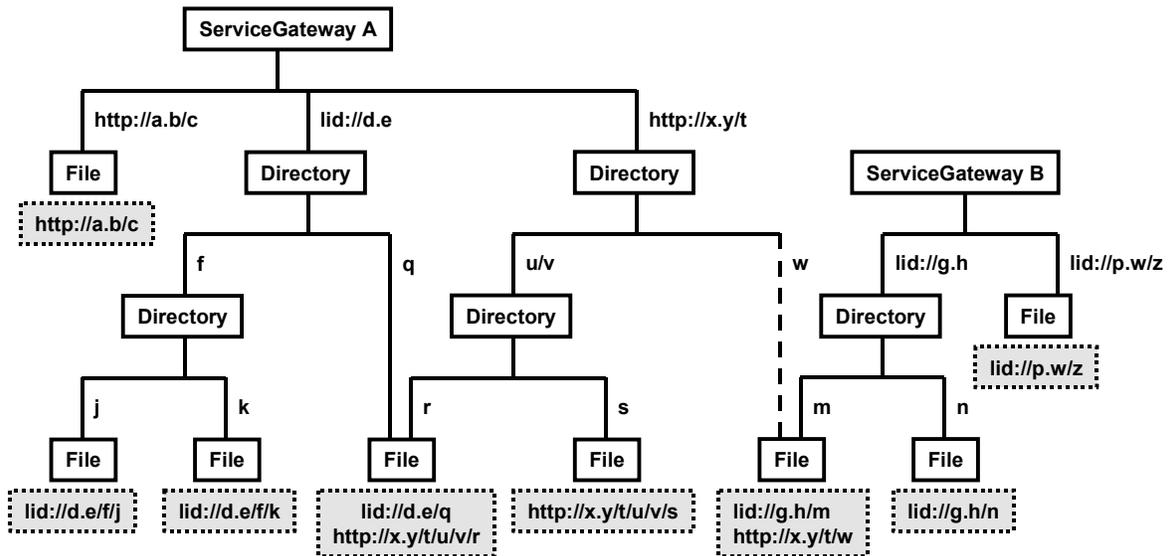


Figure B6 URIs derived from TSFS directory structure.

The following algorithm can be used to resolve a “lid:” URI into a file, using the processes given in Sections 2 and 3 of this annex for IOR resolution. It is assumed that the data service and the Data Service Table application containing the file are known. If the Data Service Table application is not known, the receiver must perform this algorithm for all applications in the Data Service Table. If the data service is not known, the receiver must scan through all the virtual channels it can receive, and perform this algorithm for every Data Service Table application of every data service it encounters until it succeeds in finding the file. This is considerably more difficult, in fact quite likely not feasible in practice.) It is assumed that there is no caching service performed; i.e., there is no TSFS service information being kept in local storage on the receiver.

Input: URI to be resolved:

- 1) Acquire the DST for the data service and check all the Taps of the relevant application for protocol_encapsulation value 0x0F. This gives a list of all the candidate File Systems.
- 2) For each Tap in the list that has a URI in its selector, restricting the scope of the Tap, check to see if the URI in the selector matches some leftmost substring of the input URI. If not, discard that Tap. If so, put that Tap at the head of the list, since it is an especially likely candidate.
- 3) For each Tap in the list, perform steps 4 through 7 below.
- 4) Use the association tag in the Tap to identify the program element that has the DSI message containing the ServiceGatewayInfo for the TSFS. Check all DSI messages that appear in that program element for one with a carouselId in the carouselNSAPAddress that matches the carouselId in the Tap selector. This is the DSI message for the TSFS.
- 5) Extract the IOR of the Service Gateway object from this DSI message.

- 6) A very easy way to handle the recursive search down the directory tree is to utilize a list in which each list item consists of a pair of components, an IOR and a character string (where the character string consists of an unmatched portion of the input URI). The initial list contains just one item, consisting of the Service Gateway IOR and the input URI.
- 7) Go down the list in order. For each item in the list, which we will designate by `this_item`, containing `this_IOR` and `this_string`, perform the following:
 - a) If the object referenced by `this_IOR` is a file object and `this_string` is an empty string, the desired file has been found. The file can be acquired, and the algorithm can be terminated.
 - b) If the object referenced by `this_IOR` is a file object and `this_string` is a non-empty string, discard the item from the list.
 - c) If the object referenced by `this_IOR` is a directory object (or a Service Gateway object), check each binding in the object to see if the binding name matches some left-most substring of `this_string`. For each match that is found, create a new list item containing the IOR associated with the binding and a new character string obtained by deleting the matched left-most substring from `this_string`. Add each such new item to the end of the list. Then discard `this_item` from the list.
 - d) If you come to the end of the list, so that there are no more items to process, then the original URI has no matching file in this TSFS. Quit and go on to the next TSFS.
- 8) If you come to the end of the list of candidate File Systems without success, then the original URI has no matching file in this data service.

Output: the requested file data:

If the receiver initially acquires and caches all the directory objects in its local store, all of these steps except for the actual retrieval of the final object go very quickly. If the receiver initially acquires and caches all the directory and file objects in its local store, it can work out the URI associated with each object and set up a URI index to the objects, similar to the index that web browsers use for their cache. Then the acquisition of any object can be achieved very quickly from the local cache, without going through the steps above.

Annex C: TSFS Objectives (Informative)

1. OBJECTIVES FROM RFP

This standard was drafted in response to the objectives listed in Table C1. (See [RFP].) The language in this table is taken verbatim from the Request for Proposals. As such, the use of “shall” is simply the language that was chosen to indicate necessary features in proposals. It is not to be interpreted as normative in the context of this standard.

An explanation of how these objectives are addressed in the standard can be found in Section 2 of this annex.

Table C1 TSFS Objectives

Conformance Requirements
The TSFS shall allow a Hierarchical name-space.
The TSFS shall preserve the reference hierarchical naming structure created by the content provider.
The TSFS shall allow files and directories to be carouseled, so that the user can browse the content at any time.
The TSFS shall allow selective downloading of data files to the receiver.
The TSFS shall not constrain the entire file hierarchy to be conveyed in a single Program Element or a single Transport Stream.
The TSFS shall allow at least the following attributes: timestamp/versioning, file size.
The underlying protocol shall accommodate carriage of delivery parameters associated with each file or directory of the TSFS.
The TSFS shall allow multiple directories to reference a single file (i.e., the delivery parameters would be same for both files, but they are listed in 2 or more directories).
The TSFS shall allow file transmission in any arbitrary order with arbitrary frequency.
The TSFS shall support versioning of files.
The TSFS shall allow the conversion of relative file names to absolute file names.
The absolute path names shall be unique in space and time.
The TSFS shall be allowed to span multiple Virtual Channels.
The TSFS shall map the “lid:” URI syntax to the file system namespace. Furthermore, multiple lid:’s may refer to a single file and the lid: name space can have multiple roots.
Every file shall have a content type label syntax as defined in RFC2046 (MIME)

2. ACHIEVEMENT OF OBJECTIVES

This standard was developed to satisfy the list of objectives presented in Section 1 of this annex. These objectives are satisfied as follows:

2.1 Name Spaces

The TSFS decouples the URI namespace from the Object Carousel’s NameServer name space. Designers of a TSFS can have a single directory one level below the Service Gateway to create a single base-URI for the entire TSFS, thereby inducing an identity mapping between the two

name spaces, or they can have multiple directories (or files) one level below the Service Gateway, thereby providing multiple base-URIs for different parts of the file system.

Each TSFS object has two names: a 3-tuple Object Location (see Section 5.3) and a URI. While the tuple name space, which satisfies objective 7, is anticipated to be useful for transport decoders, the URI name space is anticipated to be useful for application environments. The two name spaces and usage patterns are linked by a URI resolution mechanism, an example of which is presented in Annex B.

This approach enables the decoupling of transport topologies from application name spaces, virtually eliminating the ramifications of Object Carousel design constraints up-stream from the emission station. Specifically, objectives 1, 2 are preserved by the use of base-URIs. Objective 8 is satisfied by allowing multiple directories to contain copies of the same IOR.

The use of base-URIs as names for directories or files immediately below the Service Gateway and relative paths for all other directories or files was designed to meet objectives 11, 12, 14. The base-URI is used as the prefix converting relative path names to absolute URIs.

2.2 Selective Acquisition and Browsing

There is no need to acquire the entire TSFS when only a portion of the resources are needed, effectively satisfying objectives 3, 4. For example, when only one directory is needed, it is only necessary to acquire the files referenced by this directory. This is achieved by identifying the DII messages pointing to the modules in which the desired files are present, and resolving as specified in Annex B. When appropriate, the design of the physical transport of the TSFS in data carousel modules can be chosen to make this type of selectivity especially convenient.

2.3 Carriage within Virtual Channels

The TSFS specification requires the entire TSFS to be delivered in a single virtual channel, but puts no constraints on how many program elements of the virtual channel are used or how the objects are distributed among modules, or how the DSI, DII and DDB messages delivering the modules and control information are distributed among the program elements. However, [DSMCC] requires (in Sections 7.5.4 and 9.2.5) that all DDB messages of a download scenario, and therefore all modules carrying objects of a TSFS appear in a single program element, and that no DDB messages of other download scenarios can appear in that program element. There is no limit on the number of distinct TSFSs that can be delivered in a single virtual channel. Moreover, it is possible for directory entries in a TSFS to represent “soft” links (short cuts) to objects in other TSFSs, thereby effectively allowing a TSFS to span multiple virtual channels in multiple transports. This satisfies objectives 5, 13.

2.4 Optimization Opportunities

Through the decoupling of logical name spaces from the physical transport organization, the TSFS design satisfies objective 9, as well as objective 4, and thus presents numerous optimization opportunities. Additional optimization opportunities exist with respect to intelligent carouseling of modules.

2.5 Meta-Data Extensibility

The use of ObjectInfo descriptors is intended to provide extensibility. While originally designed to satisfy objectives 6, 10, 15 it enables other standards or proprietary extensions to encode generic meta-data using the descriptor framework.