
**Information technology — Coding of
audio-visual objects —**

**Part 6:
Delivery Multimedia Integration Framework
(DMIF)**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 6: Charpente d'intégration de livraison multimédia (DMIF)*

Reference number
ISO/IEC 14496-6:2000(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2000

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

Page

1	Scope	1
2	Conformance.....	1
2.1	Requirements for conformance	1
2.2	DMIF-Application Interface	1
2.3	DMIF signalling messages.....	2
3	Normative references	2
4	Terms and definitions	2
5	Abbreviations	3
6	Organization of this part of ISO/IEC 14496.....	3
7	Compliance terms.....	5
8	DMIF overview.....	5
8.1	ISO/IEC 14496 terminal architecture	5
8.2	The objectives of DMIF.....	6
8.3	The DMIF communications architecture	6
8.4	DMIF computational model.....	8
8.5	DMIF QoS model	9
8.6	The User Plane in the Delivery Layer	9
9	Remote interactive scenario.....	10
9.1	Additional requirements.....	10
9.1.1	Introduction	10
9.1.2	Billing policies.....	11
9.1.3	Possible evolution to Heterogeneous Networks	11
9.2	DMIF within the context of the ISO/OSI layers.....	11
10	The DMIF-Application Interface	12
10.1	The DMIF-Application Interface key concepts	12
10.2	Common semantic elements	13
10.2.1	channelDescriptor	13
10.3	DMIF-Application Interface primitives	15
10.4	DMIF-Application Interface semantics.....	17
10.4.1	DA_ServiceAttach ().....	17
10.4.2	DA_ServiceAttachCallback ().....	18
10.4.3	DA_ServiceDetach ().....	18
10.4.4	DA_ServiceDetachCallback ().....	18
10.4.5	DA_ChannelAdd ()	18
10.4.6	DA_ChannelAddCallback ().....	19
10.4.7	DA_ChannelDelete ()	19
10.4.8	DA_ChannelDeleteCallback ().....	19
10.4.9	DA_UserCommand ()	19
10.4.10	DA_UserCommandCallback ()	19
10.4.11	DA_Data ().....	19
10.4.12	DA_DataCallback ().....	20
10.4.13	DA_UserCommandAck()	20
10.4.14	DA_UserCommandAckCallback().....	20
10.4.15	DA_ChannelMonitor()	20
10.4.16	DA_ChannelEvent()	20
10.5	QoS Monitoring	21
10.5.1	Monitoring events.....	21

10.5.2	qosMode	21
10.5.3	qosReport	23
11	The DMIF-Network Interface	23
11.1	The DMIF-Network Interface key concepts	23
11.2	Common syntax elements	24
11.2.1	DMIF Descriptors	24
11.2.2	DMIF to DMIF data.....	26
11.2.3	Resource Descriptors.....	26
11.2.4	Direction parameter	28
11.2.5	Reason parameter.....	28
11.2.6	Response parameter.....	28
11.2.7	channelDescriptor and qosDescriptors.....	29
11.3	DMIF-Network Interface primitives.....	34
11.4	DMIF-Network Interface semantics	36
11.4.1	DN_SessionSetup ().....	36
11.4.2	DN_SessionRelease ()	36
11.4.3	DN_ServiceAttach ().....	36
11.4.4	DN_ServiceDetach ().....	37
11.4.5	DN_TransMuxSetup ()	37
11.4.6	DN_TransMuxRelease ().....	37
11.4.7	DN_ChannelAdd()	37
11.4.8	DN_ChannelAdded()	38
11.4.9	DN_ChannelDelete ()	38
11.4.10	DN_TransMuxConfig ().....	39
11.4.11	DN_UserCommand ().....	39
11.4.12	DN_UserCommandAck ()	39
12	Control Plane mappings.....	40
12.1	Default syntax.....	40
12.1.1	Syntax elements.....	40
12.1.2	DNI mapping to DMIF signalling messages	46
12.2	Syntax for IP networks with (or without) RSVP signalling, using TCP for DMIF signalling.....	56
12.2.1	Approach overview	56
12.2.2	DSM-CC Resource Descriptors used	56
12.2.3	Usage of networkSessionIds	56
12.2.4	Usage of transactionIds	57
12.2.5	DNI mapping to socket actions and RSVP signalling	57
12.3	Syntax for IP networks with (or without) RSVP Signalling, using UDP for DMIF signalling	59
12.3.1	Approach overview	60
12.3.2	DSM-CC Resource Descriptors used	60
12.3.3	Usage of networkSessionIds	60
12.3.4	Usage of transactionIds	60
12.3.5	DNI mapping to socket actions and RSVP signalling	60
12.4	Syntax for ATM networks with Q.2931 signalling	61
12.4.1	Approach overview	61
12.4.2	DSM-CC Resource Descriptors used	61
12.4.3	Usage of networkSessionIds	61
12.4.4	Usage of transactionIds	61
12.4.5	DNI mapping to Q.2931 signalling messages	62
12.5	Syntax for Networks with ITU-T H.245 Signaling	66
12.5.1	Approach overview	66
12.5.2	DSM-CC Resource Descriptors used	66
12.5.3	Usage of networkSessionIds	66
12.5.4	Usage of transactionIds	67
12.5.5	DNI mapping to H.245 Signaling messages	67
12.5.6	Data transmission procedures	68
13	Terminal Capability Matching	68
13.1	DMIF Default signalling with Compatibility Descriptors	68

Annex A (informative) Overview of DAI and DNI parameters	69
A.1 Sessions and services	69
A.2 Channels	69
Annex B (informative) Information flows for DMIF	71
B.1 Information flows for Remote Interactive DMIF	71
B.1.1 Initiation of a service in a Remote Interactive DMIF	72
B.1.2 Addition of Channels in a Remote Interactive DMIF	74
B.1.3 Deletion of Channels in a Remote Interactive DMIF	80
B.1.4 Termination of a Service in a Remote Interactive DMIF	82
B.2 Information Flows for Broadcast DMIF	83
B.2.1 Initiation of a service in a Broadcast DMIF	84
B.2.2 Addition of Channels in a Broadcast DMIF	85
B.2.3 Deletion of Channels in a Broadcast DMIF	86
B.2.4 Termination of a service in a Broadcast DMIF	87
B.3 Information Flows for Local Storage DMIF	87
B.3.1 Initiation of a service in a Local Storage DMIF	88
B.3.2 Addition of Channels in a Local Storage DMIF	89
B.3.3 Deletion of Channels in a Local Storage DMIF	90
B.3.4 Termination of a service in a Local Storage DMIF	91
Annex C (informative) Use of URLs in DMIF	92
C.1 Introduction	92
C.2 Generic concepts	92
C.3 URL schemes allowed in DMIF	92
C.4 New URL schemes	92
C.4.1 URL scheme for DMIF signalling over UDP/IP	93
C.4.2 URL scheme for DMIF signalling over TCP/IP	93
C.4.3 URL scheme for DMIF signalling over networks using NSAP addresses format	93
Annex D (informative) Protocol error recovery	94
D.1 Timouts and retransmission	94
D.2 Transaction state variables	94
D.2.1 Transaction identifier: Tid	94
D.2.2 Transaction state: Tid.state	94
D.2.3 Associated session: Tid.Sid	94
D.2.4 Transaction message timeout: Tid.tMessage	94
D.2.5 Expired transactionId holding timer: Tid.tHold	95
D.2.6 Transaction message: Tid.message	95
D.2.7 Tid.retransBound	95
D.2.8 Tid.numRetrans	95
D.3 Transaction Identifier State Machine	95
D.4 Transaction Identifier State table	96
Annex E (informative) Subset of DSM-CC resource descriptors from DSM-CC ISO/IEC 13818-6	98
E.1 ResourceDescriptor format	98
E.1.1 ResourceDescriptorDataField	98
E.1.2 Specifying ranges and lists of values in resource descriptors	99
E.2 Resource descriptor types useful to this part of ISO/IEC 14496	101
E.3 Resource descriptor definitions	102
E.3.1 AtmSvcConnection resource descriptor definition	102
E.3.2 IP resource descriptor definition	102
E.3.3 AtmVcConnection resource descriptor definition	103
Annex F (informative) ISO/IEC 14496 content carried over an ETS 300 401 system	104
F.1 Introduction	104
F.2 ISO/IEC 14496 content embedded in an ETS 300 401 multiplex	104
F.2.1 Overview of ETS 300 401 payload transmission	104
F.2.2 Object descriptor encapsulation	105
F.2.3 Scene description stream encapsulation	109
F.2.3.1 BIFSCCommandDataField	110

F.2.4	ClockReferenceStream encapsulation	111
F.2.5	Audiovisual stream encapsulation.....	111
F.2.6	ISO/IEC 14496 stream indication in ETS 300 401 Service Information	112
F.3	Walkthrough of a typical MPEG-4 session carried in an ETS 300 401 system.....	112
F.3.1	General.....	112
F.3.2	Start of a session.....	113
F.3.3	Retrieval of the InitialOD	113
F.3.4	Retrieval of the subsequent BIFS and Object Descriptor streams.....	113
F.3.5	Retrieval of the clock reference and the audio-visual streams.....	113
F.3.6	Dynamic changes to the scene	113
F.4	Mapping of DAI procedures to ETS 300 401 procedures.....	113
F.4.1	Initiation of an MPEG-4 service in a DAB system.....	114
F.4.2	Addition of channels in an MPEG-4 service in a DAB system	114
F.4.3	Deletion of Channels in an MPEG-4 service in a DAB system	115
F.4.4	Termination of a service in an MPEG-4 service in a DAB system	115
Annex G	(informative) Patent statement	116
G.1	Patent statement	116
Annex H	(informative) ISO/IEC 14496 content carried over an H.324 system	118
H.1	Introduction.....	118
H.2	Walkthrough of a typical MPEG-4 session carried in an H.324 system	118
H.2.1	General	118
H.2.2	Start of a session.....	118
H.2.3	Retrieval of the InitialOD.....	118
H.3	Mapping of DAI procedures to H.245 procedures.....	118
H.3.1	Initiation of a service in a Remote Interactive DMIF	119
H.3.2	Addition of Channels in a Remote Interactive DMIF	119
H.3.3	Deletion of Channels in a Remote Interactive DMIF	121
H.3.4	Termination of a Service in a Remote Interactive DMIF	122
Annex I	(informative) DMIF Application Programming Interface – Syntax definition	123
I.1	Objectives of this Annex	123
I.2	DAI protocol flow	123
I.3	Application Programming Interface	124
I.3.1	DMIF primitives	124
I.4	DMIF Application Programming Interface – ANSI C/C++	126
I.4.1	Reference Implementation	126

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 14496-6 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 14496-6:1999), which has been technically revised.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

- *Part 1: Systems*
- *Part 2: Visual*
- *Part 3: Audio*
- *Part 4: Conformance testing*
- *Part 5: Reference software*
- *Part 6: Delivery Multimedia Integration Framework (DMIF)*

Annexes A to I of this part of ISO/IEC 14496 are for information only.

Introduction

The digital technology has transformed the generation, storage and communication of information. With this, the information age was borne allowing humankind to generate multimedia contents, communicate them and store them in meaningful ways. Digitization has also meant a rich variety of delivery and storage technologies, each bringing its own unique characteristics, over which multimedia material is communicated, broadcast or stored. The MPEG Delivery Multimedia Integration Framework (DMIF) allows each delivery technology to be used for its unique characteristics in a way transparent to application developers. DMIF specifies the semantics for the DMIF-Application Interface (DAI) in a way that it satisfies the requirements for both broadcast, local storage and remote interactive scenarios while maintaining uniformity across all cases. By including the ability to bundle connections into sessions DMIF enables network operators to apply appropriate billing policies in the provision of multimedia services. By adopting QoS metrics which relate to the media and not to the transport mechanism, DMIF hides the delivery technology details to applications. These unique features of DMIF give multimedia application developers what they need in terms of permanence and richness beyond what is possible with each individual delivery technology. With DMIF, application developers can begin to invest in commercial multimedia applications with the assurance that their investment will not be made obsolete by new delivery technologies. In order to fully reach its goal, DMIF needs "real" instantiation of its DMIF-Application Interface, and well defined, specific mappings of DMIF concepts and parameters into existing signalling infrastructures.

Information technology — Coding of audio-visual objects —

Part 6: Delivery Multimedia Integration Framework (DMIF)

1 Scope

This part of ISO/IEC 14496 specifies the Delivery Layer of ISO/IEC 14496, which allows applications to transparently access and view multimedia streams whether the source of the streams is located on an interactive remote end-system, the streams are available on broadcast media or they are on storage media.

In this part of ISO/IEC 14496, the following aspects are covered:

DMIF communication architecture

DMIF-Application Interface (DAI) definition

URL semantic to locate and make available the multimedia streams

DMIF Default Signalling Protocol (DDSP) for remote interactive scenarios, and its related variations using existing native network signaling protocols

Information flows for player access to streams on remote interactive end-systems, from broadcast media or from storage media

2 Conformance

2.1 Requirements for conformance

DMIF conformance is only requested at the boundary of the equipment implementing a DMIF Instance. This includes the syntax and semantics of the particular Control protocol selected for a particular network (e.g., DMIF signalling messages), as well as the compliance to the mappings defined for broadcast and local storage systems.

DMIF conformance specifications is provided in ISO/IEC 14496-4.

2.2 DMIF-Application Interface

The DAI is a reference point. The exact syntax of the DAI is not defined by this part of ISO/IEC 14496, and does not represent a conformance point.

The DAI does not impose any programming language, nor syntax (e.g., the exact format for specifying a particular parameter -within the bounds of its semantic definition- or the definition of reserved values). Moreover the DAI provides only the minimal semantics for defining the behaviour of DMIF.

A real interface needs more than what is specified in the DAI (e.g.; methods to initialise, reset, reconfigure, destroy, query the status or register services). Most of these aspects, as well as the detailed syntax, deeply depend on the implementation and language bindings. Moreover such details have no impact on the DMIF model and on the conformance issues, and therefore they are out of the scope of DMIF.

2.3 DMIF signalling messages

This part of ISO/IEC 14496 specifies the syntax and semantics of a few instances of protocols for DMIF peer to DMIF peer interactivity, designed for specific network environments. A DMIF Instance making use of one of the specified protocols shall fully comply to it. A DMIF Instance implementing a protocol not specified in this International Standard cannot be tested for conformance, but may still be evaluated at the DAI Reference Point.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 14496. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 14496 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ETS 300 401:1997, *Radio broadcasting systems: Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*.

IEEE 802:1990, *IEEE Organization Unique Identifier*.

ISO/IEC 13818-6:1998, *Information technology — Generic coding of moving pictures and associated audio information — Part 6: Extension for DSM-CC*.

ISO/IEC 14496-1:—¹⁾, *Information technology — Coding of audio-visual objects — Part 1: Systems*.

ITU-T Recommendation Q.2931:1995, *Broadband Integrated Services Digital Network (B-ISDN) — Digital Subscriber Signalling System No. 2 (DSS 2) — User-Network Interface (UNI) — Layer 3 specification for basic call/connection control*.

ITU-T Recommendation E.164:1997, *The international public telecommunication numbering plan*.

RFC1738:1994, *Uniform Resource Locators (URL)*.

4 Terms and definitions

For the purposes of this part of ISO/IEC 14496, the following terms and definitions apply.

4.1 Association Tag: In the context of this specification an Association Tag is used to identify elements within a Network Session with unique end-to-end significant values.

4.2 Channel: Is the entity over which a DMIF User sends or receives data.

4.3 Delivery layer: The layer of an ISO/IEC 14496 terminal that abstracts the delivery mechanism used to store or transmit streaming data and associated control information.

4.4 DMIF-Application Interface: Is the interface between an application (DMIF User) and the Delivery layer.

4.5 DMIF Default Signalling Protocol (DDSP): A session level protocol for the management of multimedia streaming over generic delivery technologies defined in this part of ISO/IEC 14496. The protocol comprises primitives to setup and tear down sessions as well as individual data channels.

4.6 DMIF Instance: Is an implementation of the Delivery layer for a specific delivery technology.

1) To be published. (Revision of ISO/IEC 14496-1:1999)

4.7 DMIF-Network Interface: Is a semantic API that abstracts the signalling between DMIF peers irrespectively of the delivery support.

4.8 DMIF peer: a shorthand for "peer DMIF Instance".

4.9 DMIF User: Is the application that exploits the functions offered by the Delivery Layer through the DAI.

4.10 Heterogeneous Network: A Network composed of different transport technologies which are connected in tandem through InterWorking Units.

4.11 Homogeneous Network: A Network composed of one transport technology only.

4.12 Network Session: An association between two DMIF peers providing the capability to group together the resources needed for an instance of a service. The Network Session is identified by a network-wide unique ID. A Network Session could group one or more Service Sessions.

4.13 Service: Is an entity identified by a Service Name (opaque to DMIF) which responds to DAI primitives.

4.14 Service Session: A local association between the local DMIF Instance and a particular service.

4.15 Transmux Channel: A native network transport channel: it includes the transport protocol stack that is directly supported by the signalling means of the network. It provides the basic transport stack element that DMIF may complement with additional multiplexing or protection tools.

5 Abbreviations

CAT: Channel Association Tag

DAI: DMIF-Application Interface

DDSP: DMIF Default Signalling Protocol

DS: DMIF Signaling

DNI: DMIF-Network Interface

QoS: Quality of Service

TAT: Transmux Association Tag

URL: Universal Resource Locator

6 Organization of this part of ISO/IEC 14496

This part of ISO/IEC 14496 contains the following technical clauses:

Clause	Definition
8	DMIF Overview - This clause explains how DMIF reaches its two-fold goal of hiding the delivery technology details from the DMIF User, and ensuring interoperability between end-systems. It describes the DMIF communications architecture and explains how this architecture can support current and future delivery technologies, preserving application from continuous updates to follow evolving technologies.
9	The remote interactive scenario - In this clause the additional requirements due to a remote

interactive scenario are considered.

- 10 The DMIF-Application Interface - This clause defines a (semantic) API, called the DAI, that allows for the development of applications involving media content delivery, irrespective of delivery technology. The DAI provides primitives for contacting remote services, for establishing data communication channels with these services, and also exchanging data over these channels. It should be noted that the DAI provides a minimum semantics, which means that a DMIF User would have to turn to a specific implementation of the DAI for a syntax; moreover this implementation would probably provide additional functionality. The only elements whose syntax is specified at the DAI are the URL, for locating services.
- 11 The DMIF-Network Interface (informative) - The DMIF-Network Interface (DNI) is a semantic API that abstracts the signalling between DMIF peers irrespective of the delivery support. The DNI is informative, and is introduced in support to the description of the DMIF Signalling Protocol.
- 12 Control Plane mappings - This clause describes the mappings of the DNI primitives into DMIF Signalling Protocol messages. A Default DMIF Signalling Protocol is specified as well as a set of mappings into native network signalling protocols (extended through additional DMIF signalling messages, usually carried opaque to native signalling in a dedicated channel). These specifications ensure that different implementations of a DMIF Instance interoperate.
- 13 Terminal Capability Matching – Specifies how DMIF determines compatibility between DMIF terminals.
- Annex A Overview of DAI and DNI parameters (informative) - This Annex provides the glue to link concepts and parameters as defined in DAI and DNI.
- Annex B Information Flows for DMIF (informative) - This Annex defines the scenarios for remote interactive end-system, broadcast and local storage DMIF implementations.
- Annex C Use of URLs in DMIF (informative) - This Annex defines the URL schemes currently supported.
- Annex D Protocol error recovery (informative) – This Annex specifies the state machine for the transactions in the DMIF Signalling Protocol.
- Annex E Subset of DSM-CC resource descriptors from DSM-CC ISO/IEC 13818-6 (informative) – This Annex reports selected dsm-cc resource descriptors from ISO/IEC 13818-6 that are used by this part of ISO/IEC 14496.
- Annex F ISO/IEC 14496 content carried over an ETS 300 401 System (informative) – This Annex defines how ISO/IEC 14496 content can be carried over an ETS 300 401 System (DAB), and describes how to access it through the DMIF-Application Interface.
- Annex G Patent statement (informative)
- Annex H ISO/IEC 14496 content carried over an H.324 system (informative)
- Annex I DMIF Application Programming Interface – Syntax definition (informative)

7 Compliance terms

The following table defines the meaning of terms used in this part of ISO/IEC 14496:

Table 7—1 — Definition of terms used in this part of ISO/IEC 14496

Term	Definition
shall	Means that an item is required. For example, the phrase “The widget shall be blue.” means that no color other than blue is acceptable for the widget.
shall not	Means that an item is not allowed. For example, the phrase “The widget shall not be blue.” means that the color blue is not acceptable for the widget.
must	Not used. Use “shall” instead.
must not	Not used. Use “shall not” instead.
should	Means that, if possible, an item shall be used. For example, the phrase “The widget should be blue.” Means that blue is the preferred color for the widget but, if blue is not possible, other colors are acceptable.
should not	Means that, if possible, an item shall not be used. For example, the phrase “The widget should not be blue.” Means that it is preferred that the color of the widget not be blue, however it is allowable to use that color if necessary.
may	Means that an item is optional. For example, the phrase “The widget may be blue.” Means that it is allowable to use blue as the widget color.
may not	Not used. Use “shall not” instead.

8 DMIF overview

8.1 ISO/IEC 14496 terminal architecture

The generic ISO/IEC 14496 terminal architecture is depicted in Figure 1. It comprises three layers: the Compression Layer, the Sync Layer and the Delivery Layer. The Compression Layer is media aware and delivery unaware; the Sync Layer is media unaware and delivery unaware; the Delivery Layer is media unaware and delivery aware.

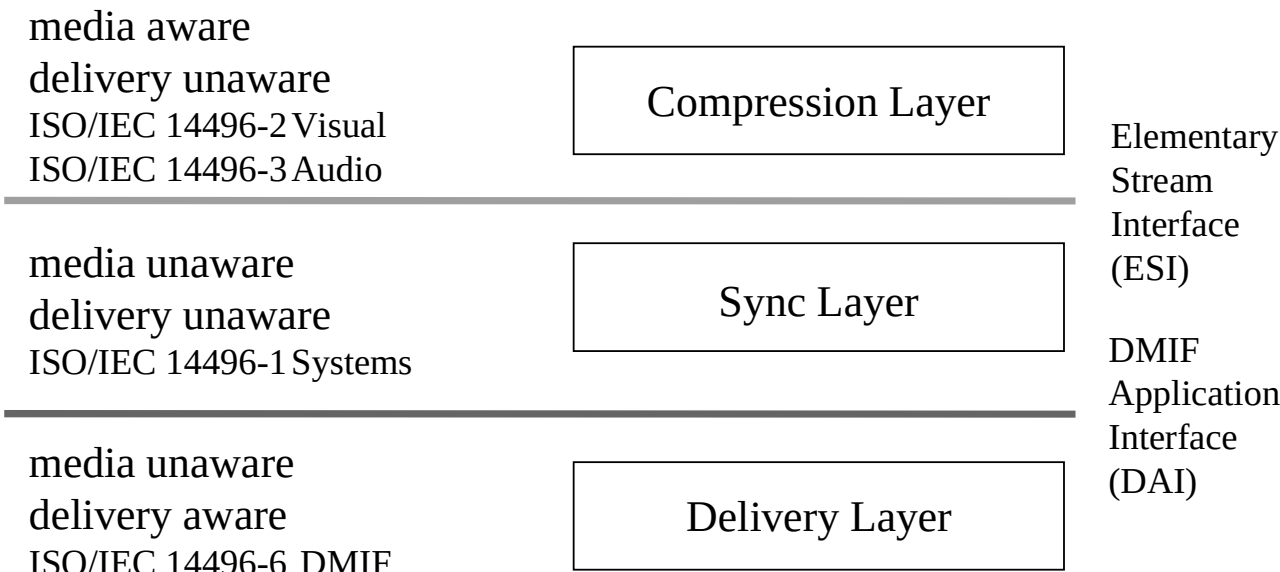


Figure 1 — ISO/IEC 14496 terminal architecture

The Compression Layer performs media encoding and decoding of Elementary Streams and is specified in ISO/IEC 14496-2:1999 and ISO/IEC 14496-3:1999; the Sync Layer manages Elementary Streams and their synchronisation and hierarchical relations and is specified in ISO/IEC 14496-1:2001; the Delivery Layer ensures transparent access to content irrespective of delivery technology and is specified in this part of ISO/IEC 14496.

The boundary between the Compression Layer and the Sync Layer is named Elementary Stream Interface (ESI) and its minimum semantic is specified in ISO/IEC 14496-1:2001.

The boundary between the Sync Layer and the Delivery Layer is named DMIF-Application Interface (DAI) and its minimum semantic is specified in this part of ISO/IEC 14496.

8.2 The objectives of DMIF

The Delivery Multimedia Integration Framework has a number of objectives:

to hide the delivery technology details from the DMIF User

to manage real time, QoS sensitive channels

to allow service providers to log resources per session for usage accounting

to ensure interoperability between end-systems

DMIF defines a communication architecture that hides the details of the delivery technologies below an interface that is exposed to the application, called the DMIF-Application Interface (DAI). Delivery technologies include transport network technologies (e.g., the Internet, or an ATM infrastructure), as well as broadcast technologies and local storage technologies.

The DAI separates the delivery aware and delivery unaware layers of the ISO/IEC 14496 terminal architecture. In the case of transport network technologies the DAI fulfils the specific needs of ISO/IEC 14496 based applications, however, its design is generic enough to allow the exploitation of DMIF in contexts other than ISO/IEC 14496 based applications.

The DAI is a semantic API that allows the development of applications irrespective of the delivery support. No syntax is specified for this Interface, since this would be programming language and operating system dependent; moreover only the minimum functionality is specified.

DMIF takes into account the QoS management aspects and the requirement of allowing service providers to log resources per session for usage accounting, in order to facilitate the implementation of appropriate billing policies. In the case of interactive networks it derives a generic protocol to fulfil the requirements (the DMIF Default Signalling Protocol – DDSP).

By adapting the generic protocol to the native network signalling mechanisms, DMIF fully specifies the behaviour of the DMIF Instances operating on particular network environments, thus ensuring interoperability for those cases. Other network environments and protocol implementations are not prevented, as far as the DAI functionality is preserved, but are out of the scope of this specification.

8.3 The DMIF communications architecture

The integration framework of DMIF covers three major technologies, interactive network technology, broadcast technology and the storage technology; this is shown in Figure 2.

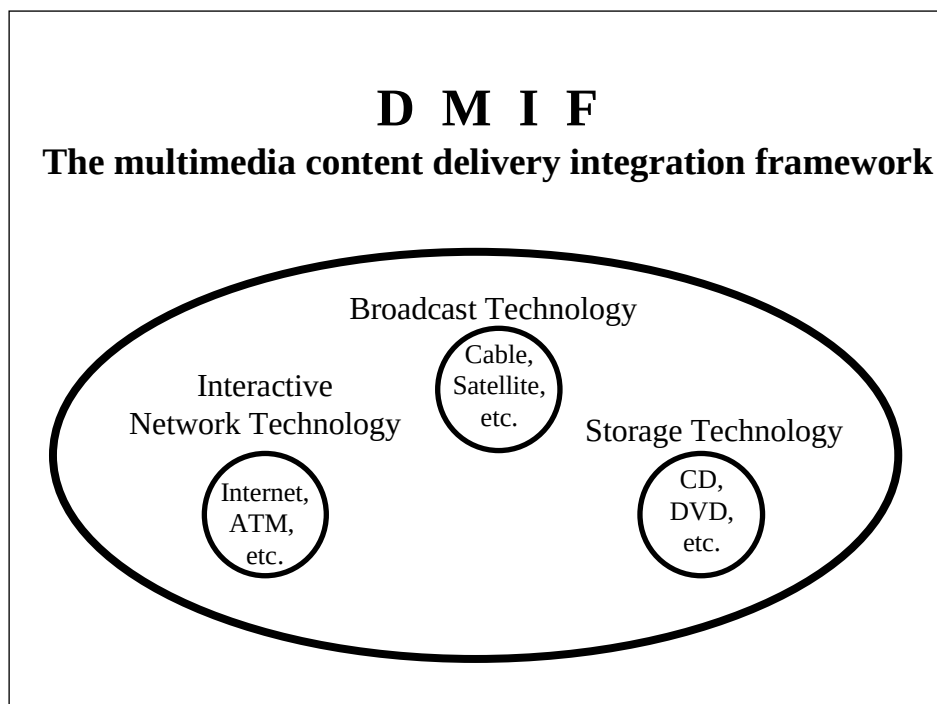


Figure 2 — DMIF addresses the delivery integration of three major technologies

Figure 3 represents the DMIF communication architecture. The shaded boxes clarify what are the boundaries of a DMIF implementation: this part of ISO/IEC 14496 normatively specifies the behaviour of a DMIF implementation at those boundaries, while the additional modelling and definitions which apply to elements internal to the shaded boxes have only informative value.

The DMIF architecture is such that applications that rely on DMIF for communications do not have to be concerned with the underlying communications method. The implementation of DMIF takes care of the delivery technology details presenting a common interface to the application.

Figure 3 depicts the above concept. An application accesses data through the DMIF-Application Interface, irrespective whether such data comes from a broadcast source, from local storage or from a remote server. A Delivery layer implementation allows the concurrent presence of more than one DMIF Instance. When requesting a specific service the Application supplies a URL that allows the Delivery layer to determine the appropriate DMIF Instance to activate. The DMIF Instances will then translate the Originating Application requests into specific actions to be taken with respect to the broadcast network or the local file system, or translate it into messages to be delivered to the Target Application, taking care of the peculiarities of the involved delivery technology. Similarly, data entering the terminal (from remote servers, broadcast networks or local files) is uniformly delivered to the Originating Application through the DAI. The DAI allows the DMIF User to group elementary streams into services and to specify the QoS requirements for the desired elementary streams.

DMIF separates the common features that should be implemented in each DMIF Instance from the details of the delivery technology. In the case of interactive networks DMIF specifies a logical interface (the DMIF-Network Interface – DNI) between a hypothetical module implementing this common features and the network specific modules. The DNI specifies the information flow that should occur between peers of such hypothetical module; the network specific modules specify the exact mapping of the DNI primitives into signalling messages.

DNI primitives need to be adapted to the native network signalling mechanisms in order to provide access to the network resources. DNI primitives represent one of the possible solutions to ensure the DAI functionality when interacting with a remote peer. A straightforward mapping of DNI primitives into signalling messages is defined, and named DMIF Default Signalling Protocol (DDSP). The DMIF Default Signalling Protocol is a session level protocol for the management of multimedia streaming over generic delivery technologies. The protocol comprises primitives to set-up and tear down sessions as well as individual data channels.

DNI primitives can be mapped into different protocols (e.g., in a LAN, or in the domain of a particular telecom operator) to provide the DAI functionality in the presence of different requirements imposed by a particular network environment (e.g., for admission control, billing policies ...).

In the case of the Broadcast and Local Storage scenarios the model is simpler, and no internal interface has been identified. Conceptually, such a DMIF Instance includes the features of a Target DMIF peer, as well as those of the Target Application. This implies that such a DMIF Instance is not agnostic of the application making use of it. Specifically, broadcast or local storage DMIF Instances have to be ISO/IEC 14496 aware if they are used in an ISO/IEC 14496 terminal. All the control messages that are exchanged between peer applications in an interactive scenario are in this case terminated in the DMIF Instance. Stream control commands like PLAY/PAUSE/MUTE are an example.

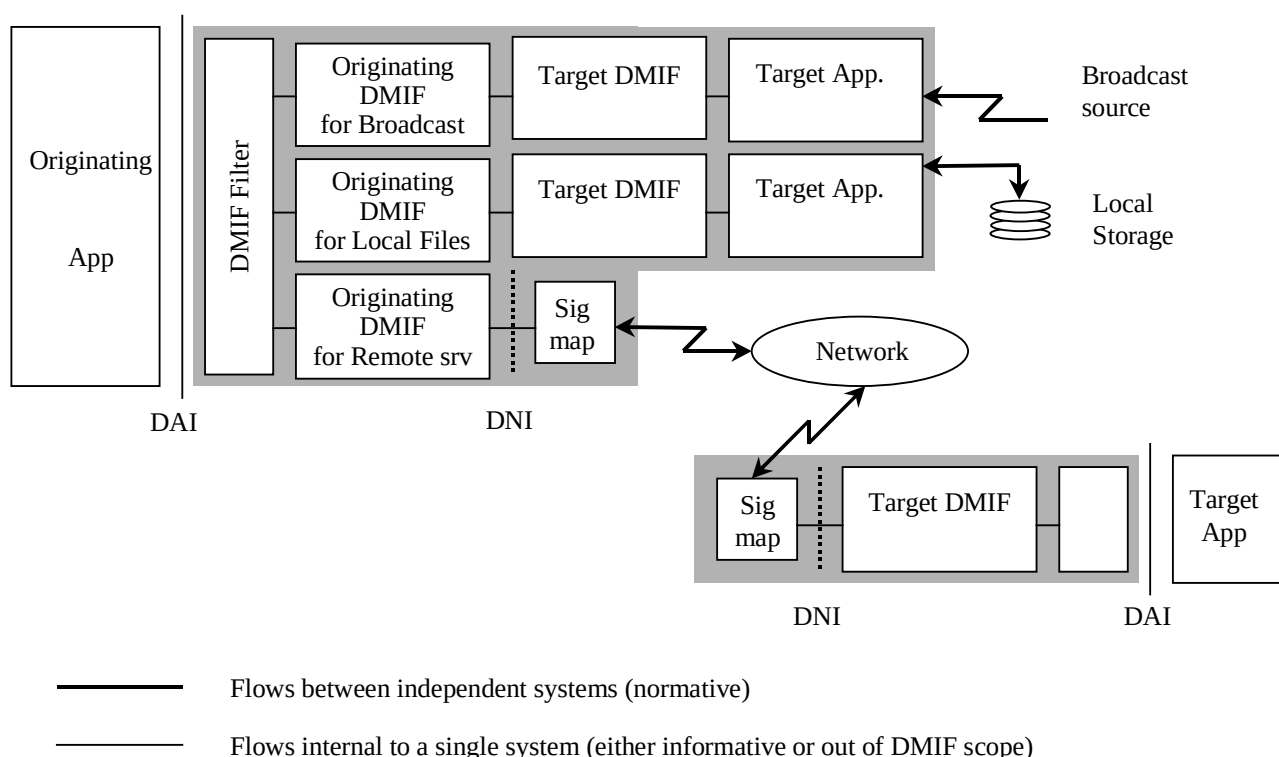


Figure 3 — DMIF communication architecture

DMIF allows the concurrent presence of one or more DMIF Instances, each one targeted for a particular delivery technology, in order to support in the same terminal multiple delivery technologies and even multiple scenarios (broadcast, local storage, remote interactive). Multiple delivery technologies may be activated by the same application, which could therefore seamlessly manage data sent by broadcast networks, local file systems and remote interactive peers.

8.4 DMIF computational model

When an application requests the activation of a service, it uses the Service primitives of the DAI (see clause 10), and creates a Service Session. In the case of a local storage or broadcast scenario, the DMIF Instance locates the content which is part of the indicated service; in case of interactive scenarios the DMIF Instance contacts its corresponding peer and creates a Network Session with it. The peer DMIF Instance in turn identifies the peer Application that runs the service, and establishes a Service Session with it. Network Sessions have network-wide significance, Service Sessions have instead local meaning. The Delivery layer maintains the association between them. Each DMIF Instance uses the native signalling mechanism for the respective network to create and then manage the Network Session (e.g.; DMIF Default Signalling Protocol integrated with ATM signalling). The application peers then use this session to create connections that are used to transport application data (e.g., ISO/IEC 14496 Elementary Streams).

When an application needs a Channel, it uses the Channel primitives of the DAI (see clause 10), indicating the Service they belong to. In the case of a local storage or broadcast scenario, the DMIF Instance locates the requested content, which is scoped by the indicated service, and prepares itself to read it and pass it in a channel to the application; in case of interactive scenarios the DMIF Instance contacts its corresponding peer to get access to the content, reserves the network resources (e.g.; connections) to stream the content, and prepares itself to read it and pass it in a channel to the application; in addition, the remote application locates the requested content, which is scoped by the indicated service. DMIF uses the native signalling mechanism for the respective network to reserve the network resources. The remote application then uses these resources to deliver the content.

Figure 4 provides a high level view of a service activation and of the beginning of data exchange in the case of interactive scenarios; the high level walk-through consists of the following steps:

The Originating Application requests the activation of a service to its local DMIF Instance: a communication path between the Originating Application and its local DMIF peer is established in the control plane (1)

The Originating DMIF peer establishes a Network Session with the Target DMIF peer: a communication path between the Originating DMIF peer and the Target DMIF Peer is established in the control plane (2)

The Target DMIF peer identifies the Target Application and forwards the service activation request: a communication path between the Target DMIF peer and the Target Application is established in the control plane (3)

The peer Applications create channels (requests flowing through communication paths 1, 2 and 3). The resulting channels in the user plane (4) will carry the actual data exchanged by the Applications.

DMIF is involved in all four steps above.

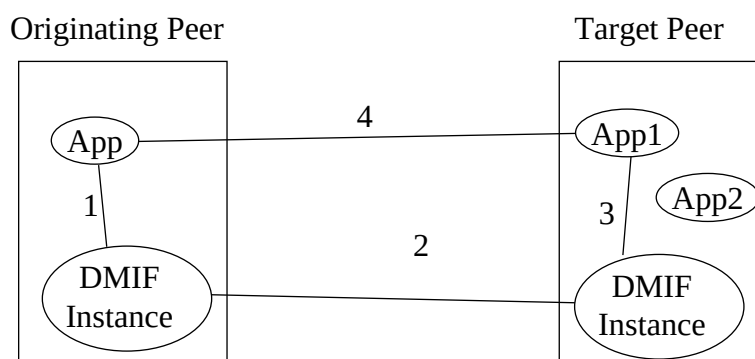


Figure 4 — DMIF computational model

8.5 DMIF QoS model

This part of ISO/IEC 14496 while it specifies the QoS traffic parameters for a given stream at the DAI (e.g., bitrate, maximum access unit size, ...), it does not specify its QoS performance requirements (e.g., delay, loss probability, ...). As a result, the QoS performance should be based only on the transport network and the administrative policy imposed by a given implementation, e.g., choose best-effort for all channels, or guaranteed service for all channels.

8.6 The User Plane in the Delivery Layer

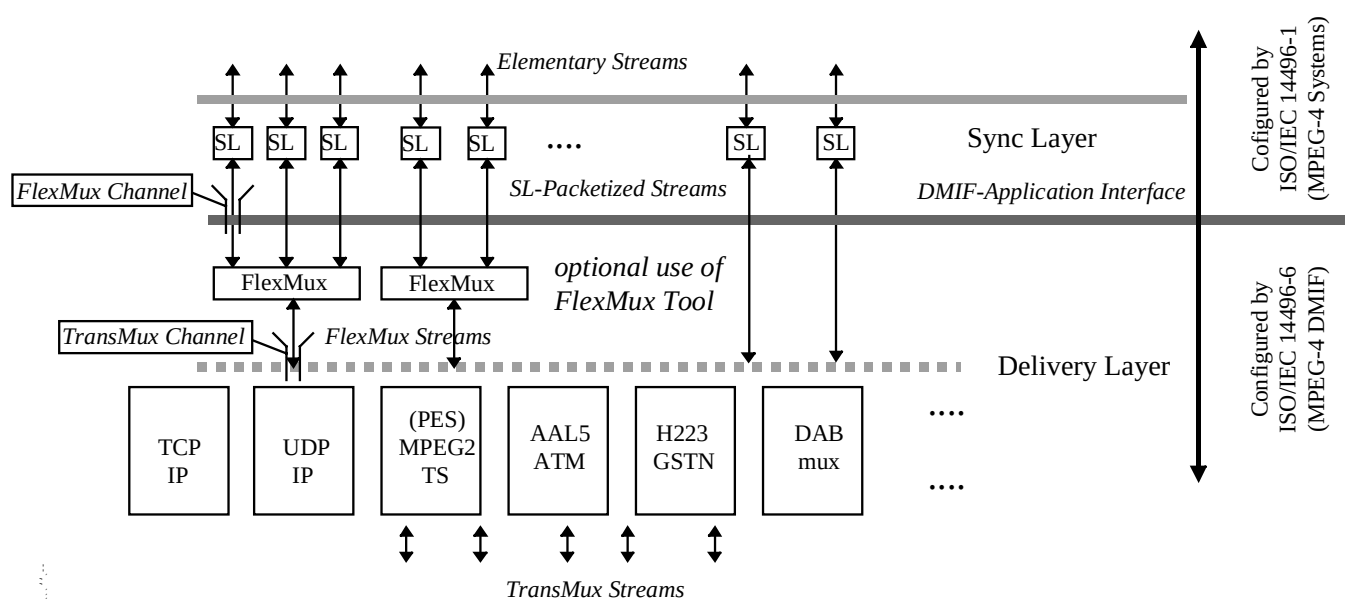
Figure 5 illustrates the User Plane in an ISO/IEC 14496 terminal, and specifically in the Delivery Layer: Elementary Streams cross the DAI in individual channels, and are multiplexed/demultiplexed in the Delivery Layer. The Delivery Layer makes use of the available multiplexing facilities of the native protocol stacks (e.g., IP port numbers, ATM VCs, MPEG-2 PIDs, file names in a file system), which are referred as Transmuxes in this specification. Where

necessary it makes use of additional multiplexing or protection tools, and uses DMIF descriptors to appropriately configure the stacks at both ends.

Currently the FlexMux multiplexing tool specified in ISO/IEC 14496-1:2001 is supported.

The Delivery layer is responsible for the configuration of the transport protocol stacks. Each DMIF Instance is in charge of configuring the exact protocol stack for each channel, and of keeping track of the associations of channels and transport resources. The configuration of the Transmux portion of the protocol stack is achieved through the usage of DMIF signalling with Resource Descriptors and/or their execution using native signalling; the configuration of the remaining portion of the protocol stack is achieved through the usage of DMIF signalling with DMIF Descriptors.

Figure 5 provides a sample of the choice of native transport protocol stacks. It outlines the fact that either an Elementary Stream or a group of streams multiplexed together (e.g., with the FlexMux tool) can be carried over a native transport.



NOTE The figure shows a number of relevant protocol stacks, that are, however, not all fully specified in this part of ISO/IEC 14496.

Figure 5 — The User Plane in a ISO/IEC 14496 terminal

9 Remote interactive scenario

9.1 Additional requirements

9.1.1 Introduction

This subclause only focuses on the Remote Interactive scenario, and defines the common features that each DMIF Instance for such a scenario should implement. These common features take care of the requirement of facilitating the possible evolution to Heterogeneous Networks and of allowing service providers to log resources per session for usage accounting, in order to facilitate the implementation of appropriate billing policies. Valid DMIF Instances may however ignore these requirements.

9.1.2 Billing policies

In order to facilitate the implementation of appropriate billing policies, DMIF allows to group in Network Sessions the resources consumed for the delivery of a certain service. This way network service providers may log and then charge the resources consumed in the course of a Network Session as a whole, instead of just summing the cost for each individual resource. Network Sessions may include multiple Service Sessions, as far as they relate to the same DMIF peers. These Service Sessions may relate to different application instances on one or both sides. This case is depicted in Figure 6. Valid DMIF Instances may however prefer to maintain a one-to-one relation between the services as requested by the DMIF User and the Network Sessions. In this case if extra capacity is available on one connection belonging to one Service Session it cannot be used by the other Service Session and as a result the end user will pay more.

Billing policies are out of the scope of this specification.

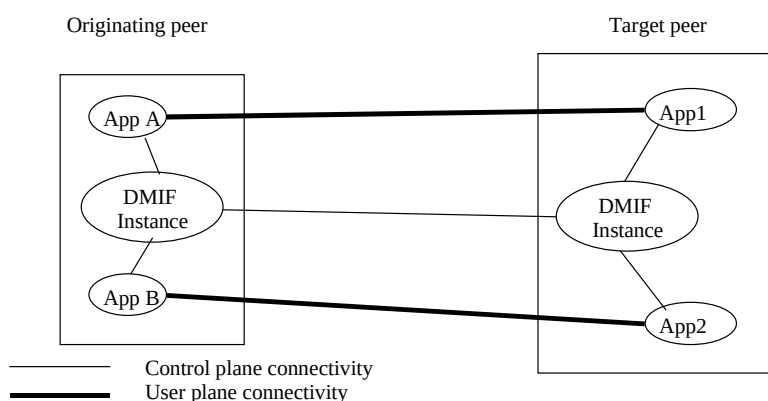


Figure 6 — A Network Session shared by multiple Service Sessions

9.1.3 Possible evolution to Heterogeneous Networks

This specification only targets homogeneous networks; however, possible evolution to Heterogeneous Networks has been considered. In order to facilitate this evolution, parts of a solution framework that was developed in ISO/IEC 13818-6 (Digital Storage Media Command & Control, User to Network – DSMCC-UN) have been adopted. In particular, the DMIF Default Signalling Protocol defined in this specification makes use of a generic descriptor architecture derived from ISO/IEC 13818-6 for describing and tagging the network resources being used.

9.2 DMIF within the context of the ISO/OSI layers

Figure 7 positions the DMIF-Application Interface in the DMIF architecture shown in Figure 3 and its networked component shown in Figure 4, and highlights the role of Delivery Layer with respect to the ISO/OSI layers: a DMIF Instance performs Session Layer functions, and the DMIF-Application Interface corresponds to a Session Service Access Point. The DMIF control primitives at the DMIF-Application Interface capture parameters that are maintained for the duration of the contract. Any other parameter that varies during the life of the contract is carried on the User Plane and is out of the scope of this specification. In the case of ISO/IEC 14496 refer to ISO/IEC 14496-1:2001. One such example of the first kind is the QoS of the media stream that is fixed during the contract and therefore carried in the DMIF control primitives. In the case of ISO/IEC 14496 an example of the second kind is the SL packet header length which could vary during the life of the media stream and is therefore carried on the User Plane as specified in ISO/IEC 14496-1:2001. Also while the DMIF control primitives are specified both at the receive and transmit ends, the User Plane is only specified at the receive end.

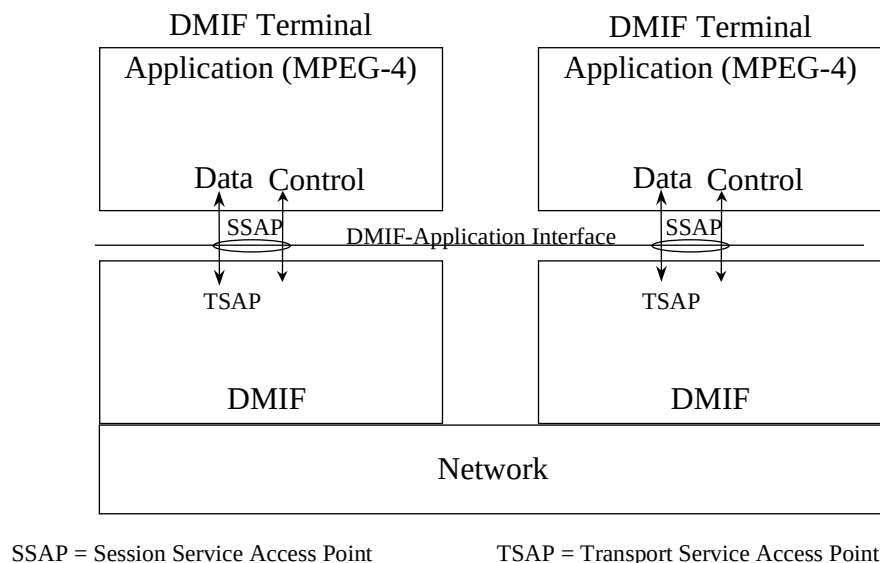


Figure 7 — Position of the DMIF-Application Interface in the DMIF architecture

10 The DMIF-Application Interface

10.1 The DMIF-Application Interface key concepts

The DMIF-Application Interface (DAI) is a semantic API that allows the development of applications transparent to the supported delivery technologies. By using the DAI, an application could seamlessly access content from broadcast networks, from local storage devices and from remote end-systems.

The DAI semantics is normative.

DAI does not impose any programming language, nor syntax (e.g., the exact format for specifying a particular parameter -within the bounds of its semantic definition- or the definition of reserved values), except for the URL defined in Annex C.. Moreover the DAI provides only the minimal semantics for defining the behaviour of DMIF.

A real implementation of the DAI needs more than what is specified here (e.g.; methods to initialise, reset, reconfigure, destroy, query the status, register services ...). Most of these aspects, as well as the detailed syntax, deeply depend on the language binding and on the implementation (part of an operating system, or a separate library, or a set of separate libraries, or a combination of the above, or ...; and for the syntax: synchronous or asynchronous implementation, callback functions or events or a polling technique, ...). Moreover such details have no impact on the DMIF model, and are therefore out of the scope of DMIF.

It is worth mentioning that this interface is available to any application, not just ISO/IEC 14496-1:2001 (MPEG-4 System) applications, i.e., the DMIF-Application Interface is designed for generic use.

The DMIF-Application Interface defines the functions offered by the Delivery layer. The entity that uses this interface is named the DMIF User.

Through the DMIF-Application Interface DMIF Users are hidden from the delivery technology details (for both Data and Control Planes), and just manipulate Service Sessions and channels.

The DMIF-Application Interface is comprised of the following classes of primitives:

Service primitives, which deal with the Control Plane, and allow the management of Service Sessions (attach and detach);

Channel primitives, which deal with the Control Plane, and allow the management of channels (add and delete);

Data primitives, which deal with the User Plane, and serve the purpose of transferring data through channels.

Note that since only the minimum DAI semantics is specified, additional primitives and parameters may be required in actual implementations, e.g., to initialize the DMIF Instance, to register callbacks, to resolve service names, to apply sophisticated statistical multiplexing at the FlexMux Layer on the sender side (including the usage of unequal protection schemes), etc.

10.2 Common semantic elements

10.2.1 channelDescriptor

DMIF defines a ChannelDescriptor at the DMIF-Application Interface to carry all informations which relate to the channel configuration. This includes in particular Quality of Service parameters for both profile (e.g. loss, delay, priority, etc.) and bandwidth requirements (e.g., peak bit rate, average bit rate, etc.) for an individual Elementary Stream. The ChannelDescriptor may also include information specific to a particular application. Two of these application dependent descriptors are defined in this document. These descriptors are intended to fully support MPEG-4 Systems based applications on any kind of delivery technology (e.g., MPEG-2 Transport Stream, RTP/RTCP or RTSP in the control plane). The information carried through the DAI in the ChannelDescriptor can be further delivered across the DNI and be mapped in the DMIF Default Signaling Protocol (DDSP).

Only the semantics of the ChannelDescriptor used at the DAI is specified in this clause.

The exact syntax of the ChannelDescriptor used at the DNI and in the DMIF Default Signaling Protocol (DDSP) is specified in subclause 11.2.4.

DMIF enables the aggregation of multiple Elementary Streams, which share the same QoS profile, into a single TransMux, and computes the aggregate bandwidth. DMIF then maps the QoS requirements (i.e., QoS profile and aggregated bandwidth requirements) for a particular TransMux into specific network QoS (e.g. traffic contract). This is shown in Figure 8.

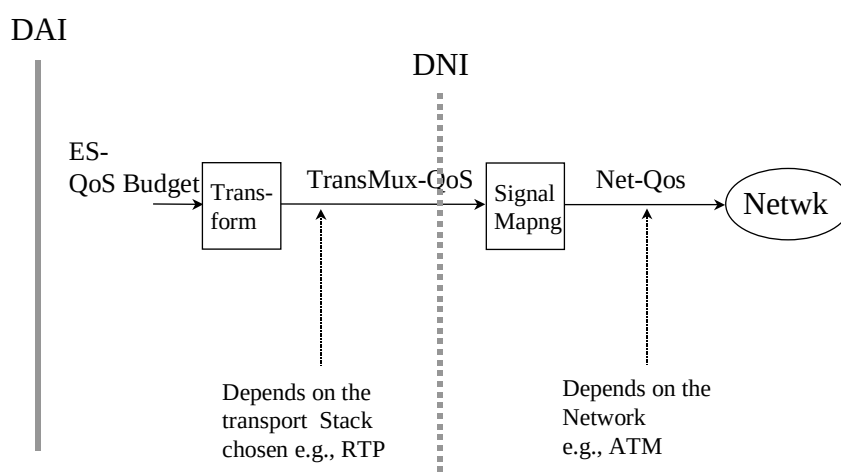


Figure 8 — Transformations of the Media QoS

In MPEG-4 the QoS parameters for an individual Elementary Stream are transmitted as part of the ES_Descriptor as defined in MPEG-4 Systems ISO/IEC 14496-1:2001. These QoS parameters represent the total media QoS budget (total-QoS). The total-QoS differs from the QoS exposed at the DAI (DAI-QoS) in that the application is expected to compute the DAI-QoS taking into account the QoS requirements and the performance of the receiver itself (decoder-QoS). This is shown in Figure 9 and Figure 10 below.

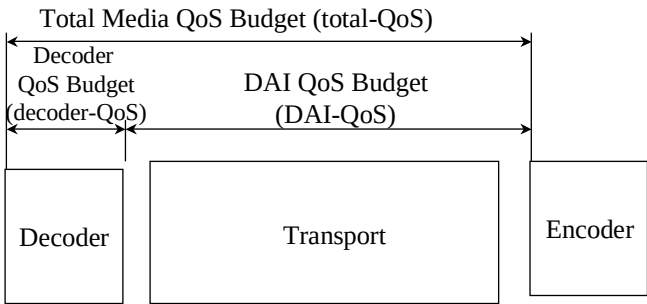


Figure 9 — Apportionment of the total end-to-end QoS Budget

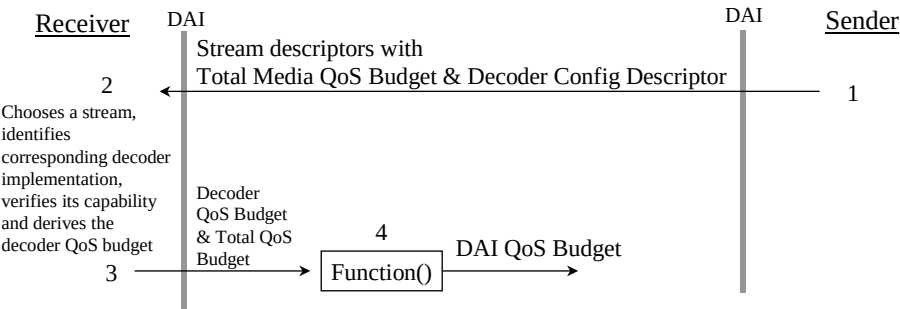


Figure 10 — The Life cycle of the Media QoS

The channel Descriptor shall be able to carry a number of QoS metrics. The set of metrics currently defined and their semantic is summarized in Table 10—1.

Table 10—1 — ISO/IEC 14496-6 defined channelDescriptor parameters.

channelDescriptor Parameter	Semantic Description
PRIORITY	Priority for the stream
MAX_AU_SIZE	Maximum size of a PDU, as delivered through the DAI
AVG_BITRATE	Average bit rate measured over an observation time window
MAX_BITRATE	Maximum bit rate measured over an observation time window
MAX_DELAY	Maximum delay experienced by any PDU of an Elementary Stream measured over an observation time window
AVG_DELAY	Average delay experienced by the PDUs of an Elementary Stream measured over an observation time window
LOSS_PROBABILITY	Allowable probability of loss of any single PDU of an Elementary stream measured over an observation time window
JITTER_TOLERANCE	Maximum delay variation experienced by any PDU of an Elementary Stream measured over an observation time
SYNC_GROUP	Identifies the timeline associated to a channel
SL_CONFIG_HEADER	Describes the MPEG-4 Sync Layer Packet Header format

The semantic meaning of the channel qualifiers tags are as follows :

PRIORITY : this qualifier defines a relative measure for the priority of an elementary stream. An elementary stream with a higher priority is more important than one with a lower priority.

MAX_PDU_SIZE : this qualifier defines the maximum size of any PDU that can be delivered through the DAI.

AVG_BITRATE : this qualifier defines the average bitrate of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

MAX_BITRATE : this qualifier defines the maximum bitrate of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

MAX_DELAY : this qualifier defines the maximum delay experienced by any PDU of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

AVG_DELAY : this qualifier defines the Average delay experienced by the PDUs of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

LOSS_PROBABILITY: this qualifier defines the allowable probability of loss of any single PDU of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

JITTER_TOLERANCE: this qualifier defines the maximum delay variation experienced by any PDU of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

SYNC_GROUP : this channel qualifier provides an identifier for the timeline associated to a channel. It allows to expose to a dmif instance the knowledge that a group of streams is sharing the same timeline.

SL_CONFIG_HEADER : this channel qualifier describes the MPEG-4 Sync Layer Packet Header format. This qualifier is comprised by a binary sequence of bytes called slConfig, whose syntax corresponds to SLConfigDescriptor defined in ISO/IEC 14496-1:2001. The length of the slConfig is specified by slConfigLen. This qualifier is therefore comprised of two semantic elements. It allows to expose to a dmif instance the knowledge that a certain channel carries content which is packetized according to the semantic rules of the MPEG-4 SL.

10.3 DMIF-Application Interface primitives

This subclause lists the primitives specified for the DMIF-Application Interface, and describes the parameters used. The only parameter with fixed syntax is the URL defined in Annex C. The C++ like formalism used to describe the primitives only aims at capturing their semantic meaning. IN and OUT keywords allow to clearly distinguish the parameters provided to the other side of the interface from those returned from it; by no means they are meant to force a synchronous implementation. The loop() construct allows to concisely represent an array of elements.

The primitives are:

- DA_ServiceAttach (IN: parentServiceSessionId, URL, uuDataInBuffer, uuDataInLen; OUT: response, serviceSessionId, uuDataOutBuffer, uuDataOutLen)
- DA_ServiceAttachCallback (IN: serviceSessionId, serviceName, uuDataInBuffer, uuDataInLen; OUT: response, uuDataOutBuffer, uuDataOutLen)
- DA_ServiceDetach (IN: serviceSessionId, reason; OUT: response)
- DA_ServiceDetachCallback (IN: serviceSessionId, reason; OUT: response)
- DA_ChannelAdd (IN: serviceSessionId, loop(channelDescriptor, direction, uuDataInBuffer, uuDataInLen); OUT: loop(response, channelHandle, uuDataOutBuffer, uuDataOutLen))

- DA_ChannelAddCallback (IN: serviceSessionId, loop(channelHandle, channelDescriptor, direction, uuDataInBuffer, uuDataInLen); OUT: loop(response, uuDataOutBuffer, uuDataOutLen))
- DA_ChannelDelete (IN: loop(channelHandle, reason); OUT: loop(response))
- DA_ChannelDeleteCallback (IN: loop(channelHandle, reason); OUT: loop(response))
- DA_UserCommand (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle))
- DA_UserCommandCallback (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle))
- DA_UserCommandAck (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle); OUT: response, uuDataOutBuffer, uuDataOutLen)
- DA_UserCommandAckCallback (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle); OUT: response, uuDataOutBuffer, uuDataOutLen)
- DA_ChannelMonitor (IN: channelHandle, qosMode; OUT: response)
- DA_ChannelEvent (IN: channelHandle, mode, qosReport)
- DA_Data (IN: channelHandle, streamDataBuffer, streamDataLen)
- DA_Data (IN: channelHandle, streamDataBuffer, streamDataLen, appDataBuffer, appDataLen)
- DA_DataCallback (IN: channelHandle, streamDataBuffer, streamDataLen, errorFlag)
- DA_DataCallback (IN: channelHandle, streamDataBuffer, streamDataLen, appDataBuffer, appDataLen, errorFlag)

The parameters have the following semantics:

appDataBuffer: is the data conveying application information. In the case of MPEG-4 this data shall be used to convey SYNC layer information.

appDataLen: is the length of the appDataBuffer field.

channelDescriptor: is a parameter set by the DMIF User containing the complete description of the Quality of Service requested for a particular channel as well as possibly application specific descriptors. Its semantic definition is given in subclause 10.2.1.

channelHandle: is a local identifier that uniquely identifies a channel in the application space, no matter how many services the application attaches to, or how many DMIF Instances it is using. In this interface specification the channelHandle parameter is set by the DMIF Instance, however it would be also acceptable if it were set by the DMIF User. The algorithm used by the DMIF Instance (User) to set the channelHandle is a matter that does not affect this interface.

direction: Indicates the direction of the channel, either UPSTREAM i.e., from the receiver to the sender or DOWNSTREAM i.e., from the sender to the receiver.

errorFlag: is a flag that indicates whether an error has been detected (but not corrected) on the streamDataBuffer.

mode: this parameter, also contained in the **qosMode** structure, indicates the type of QoS mode. The set of modes is summarized in subclause 10.5.2.

parentServiceSessionId: is a local identifier that uniquely identifies the Service Session whose URL is to be possibly used to expand the relative URL of a newly requested Service Session.

qosMode: this is a parameter set by the DMIF User containing the requested QoS Monitoring mode for a particular stream (i.e., channelHandle), as well as, associated commands with the monitoring process (i.e., start, stop, single report).

qosReport: is a parameter set by the DMIF Layer containing the QoS report for a particular stream (i.e., channelHandle). This parameter shall contain the QoS profile parameters contained in the **qosDescriptor** parameter sent in the addition of the channel (i.e., delay and loss), as well as, additional statistical parameters to be defined in the future.

reason: a code identifying the reason.

response: a code identifying the response.

serviceName: at the target DMIF peer it identifies the actual service.

serviceSessionId: is a local identifier that uniquely identifies a Service Session in the application space, no matter how many services the application attaches to, or how many DMIF Instances it is using. In this interface specification the serviceSessionId parameter is set by the DMIF Instance, however it would be also acceptable if it were set by the DMIF User. The algorithm used by the DMIF Instance (User) to set the serviceSessionId is a matter that does not affect this interface.

streamDataBuffer: is the actual Data Unit generated by the DMIF User.

streamDataLen: is the length of the streamDataBuffer field.

URL: within DMIF is a string that identifies a Service. Refer to Annex C for more information on the usage of URLs in DMIF, including the list of allowed URL schemes.

uuDataInBuffer: is an opaque structure providing upper layer information; it is transparently transported through DMIF from the local peer to the remote peer.

uuDataInLen: is the length of the uuDataInBuffer field.

uuDataOutBuffer: is an opaque structure providing upper layer information; it is transparently transported through DMIF from the remote Peer to the local Peer.

uuDataOutLen: is the length of the uuDataOutBuffer field.

10.4 DMIF-Application Interface semantics

10.4.1 DA_ServiceAttach ()

DA_ServiceAttach (IN: parentServiceSessionId, URL, uuDataInBuffer, uuDataInLen; OUT: response, serviceSessionId, uuDataOutBuffer, uuDataOutLen)

This primitive is issued by a DMIF User to request the initialization of a Service Session: the service is unambiguously identified by its **URL** which conveys information for identifying both the delivery technology being used (i.e. protocol), the address of the target DMIF peer (this may have different meanings in the different scenarios) and the name of the service inside the domain managed by the target DMIF peer -which is then referred to as **serviceName**. The **URL**, if relative, is expanded using as base URL the URL associated to the **parentServiceSessionId**.

The DMIF User might provide additional information such as client credentials in **uuDataIn**: this additional information is opaque to the Delivery layer and is only consumed by the target DMIF User (which is locally emulated in Broadcast and Local Storage scenarios).

The target DMIF User might in turn provide additional information in **uuDataOut**: this additional information is opaque to the Delivery layer and is only consumed by the local DMIF User. In an ISO/IEC 14496 application the **uuDataOut** shall return a single Object Descriptor or Initial Object Descriptor if required by the context of this call.

In case of a positive **response**, the **serviceSessionId** parameter contains the Service Session identifier that the DMIF User should refer to in subsequent interaction through the DAI regarding this Service Session.

10.4.2 DA_ServiceAttachCallback ()

DA_ServiceAttachCallback (IN: serviceSessionId, serviceName, uuDataInBuffer, uuDataInLen; OUT: response, uuDataOutBuffer, uuDataOutLen)

This primitive is issued by the target DMIF Instance to the appropriate target DMIF User as identified through the **serviceName** field. The steps involved in identifying the appropriate DMIF User and delivering the primitives to it are out of the scope of this specification.

The target DMIF Instance also provides the **serviceSessionId** parameter, that contains the Service Session identifier that the target DMIF User should refer to in subsequent interaction through the DAI regarding this Service Session.

The target DMIF User (the Application Executive running the service) might also receive additional information (e.g.; client credentials) through the **uuDataIn** field.

The target DMIF User might in turn provide additional information in **uuDataOut**: this additional information is opaque to the Delivery layer and is only consumed by the local DMIF User. In an ISO/IEC 14496 application the **uuDataOut** shall return a single Object Descriptor or Initial Object Descriptor if required by the context of this call.

In case of a negative **response**, the **serviceSessionId** becomes invalid at the target DMIF Instance.

Real implementations of this primitive are likely to support an additional parameter identifying the calling peer.

10.4.3 DA_ServiceDetach ()

DA_ServiceDetach (IN: serviceSessionId, reason; OUT: response)

This primitive is issued by a DMIF User to request the termination of the service identified by **serviceSessionId**; a **reason** should be specified. The DMIF Instance returns a **response**.

10.4.4 DA_ServiceDetachCallback ()

DA_ServiceDetachCallback (IN: serviceSessionId, reason; OUT: response)

This primitive is issued by the target DMIF Instance to inform the target DMIF User that the service identified by **serviceSessionId** has been terminated due to the reason **reason**. The target DMIF User returns a **response**.

10.4.5 DA_ChannelAdd ()

DA_ChannelAdd (IN: serviceSessionId, loop(channelDescriptor, direction, uuDataInBuffer, uuDataInLen); OUT: loop(response, channelHandle, uuDataOutBuffer, uuDataOutLen))

This primitive is issued by a DMIF User to request the addition of one or more end-to-end channels in the context of a particular Service Session identified by **serviceSessionId**.

Each channel is requested by providing an (optional) **channelDescriptor** and a **direction**

The local DMIF User might provide additional information for each requested channel in **uuDataIn**. This additional information is opaque to the Delivery layer and is only consumed by the target DMIF User. In the case of an ISO/IEC 14496 application the **uuDataIn** shall always be present.

For each requested channel, in case of a positive **response**, the **channelHandle** parameter contains the channel identifier that the DMIF User should refer to in subsequent interaction through the DAI involving this channel.

10.4.6 DA_ChannelAddCallback ()

DA_ChannelAddCallback (IN: serviceSessionId, loop(channelHandle, channelDescriptor, direction, uuDataInBuffer, uuDataInLen); OUT: loop(response, uuDataOutBuffer, uuDataOutLen))

This primitive is issued by the target DMIF Instance to the appropriate target DMIF User as identified through the **serviceSessionId** field, to inform the target DMIF User that the addition of channels is requested.

For each requested channel, the target DMIF Instance provides the **direction** to the target DMIF User. It also provides the **channelHandle** parameter, that contains the channel identifier that the target DMIF User should refer to in subsequent interaction through the DAI involving this Channel.

The target DMIF User (the Application Executive running the service) might also receive additional information through the **uuDataIn** field. In the case of an ISO/IEC 14496-1:2001 based application the **uuDataIn** shall always be present and may contain the Elementary Stream Identifier -ES_ID-.

For each requested channel, the target DMIF User returns a **response**. In case of a negative **response**, the **channelHandle** becomes invalid at the target DMIF Instance.

10.4.7 DA_ChannelDelete ()

DA_ChannelDelete (IN: loop(channelHandle, reason); OUT: loop(response))

This primitive is issued by a DMIF User to delete one or more channels as identified by **channelHandle**; a **reason** should be specified. The channels need not be all part of a single Service Session. The DMIF Instance returns a **response**.

10.4.8 DA_ChannelDeleteCallback ()

DA_ChannelDeleteCallback (IN: loop(channelHandle, reason); OUT: loop(response))

This primitive is issued by the target DMIF Instance to inform the target DMIF User that the channels identified by **channelHandle** have been closed due to the reason **reason**. The target DMIF User returns a **response**.

10.4.9 DA_UserCommand ()

DA_UserCommand (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle))

This primitive is issued by a DMIF User to send **uuData** that refers to channels identified by **channelHandle**. This primitive is intended to support the delivery of control information in the upstream direction.

10.4.10 DA_UserCommandCallback ()

DA_UserCommandCallback (IN: uuDataInBuffer, uuDataInLen loop(channelHandle))

This primitive is issued by the target DMIF Instance to inform the target DMIF User that there is **uuData** relative to channels identified by **channelHandle**.

10.4.11 DA_Data ()

DA_Data (IN: channelHandle, streamDataBuffer, streamDataLen)

DA_Data (IN: channelHandle, streamDataBuffer, streamDataLen, appDataBuffer, appDataLen)

This primitive is issued by a DMIF User to send **streamData** in the channel identified by **channelHandle**.

The second form of this primitive is issued by a DMIF User to send **streamData** in the channel identified by **channelHandle** along with application specific **appData** describing the **streamData**. In the case of ISO/IEC 14496-1:2001 based applications **appData** would carry sync layer information associated to the **streamData**.

10.4.12 DA_DataCallback ()

DA_DataCallback (IN: channelHandle, streamDataBuffer, streamDataLen, errorFlag)

DA_DataCallback (IN: channelHandle, streamDataBuffer, streamDataLen, appDataBuffer, appDataLen, errorFlag)

This primitive is issued by the DMIF Instance to the appropriate DMIF User (identified through the **channelHandle**) and provides the **streamData** along with an **errorFlag** for that channel.

The second form of this primitive is issued by the DMIF Instance to send **streamData** in the channel identified by **channelHandle** along with application specific **appData** describing the **streamData**. In the case of ISO/IEC 14496-1:2001 based applications **appData** would carry sync layer information associated to the **streamData**.

10.4.13 DA_UserCommandAck()

DA_UserCommandAck (IN: uuDataInBuffer, uuDataInLen, loop(channelHandle) ; OUT : response, uuDataOutBuffer, uuDataOutLen)

This primitive is issued by a DMIF User to send **uuDataIn** that refers to channels identified by **channelHandle**. This primitive is intended to support the delivery of control information with acknowledgement (and possibly additional information in **uuDataOut**).

10.4.14 DA_UserCommandAckCallback()

DA_UserCommandAckCallback (IN: uuDataInBuffer, uuDataInLen loop(channelHandle) ; OUT : response, uuDataOutBuffer, uuDataOutLen)

This primitive is issued by the target DMIF Instance to inform the target DMIF User that there is **uuDataIn** relative to channels identified by **channelHandle**; the DMIF User shall acknowledge the receipt of the command, possibly providing additional information in **uuDataOut**.

10.4.15 DA_ChannelMonitor()

DA_ChannelMonitor (IN: channelHandle, qosMode; OUT: response)

This primitive is issued by a user to request the initialisation, termination or single report request of a QoS monitoring process by the DMIF layer. The **channelHandle** identifies the stream to be monitored by the DMIF layer. The mode of the QoS monitoring (see subclause 10.5.2) is set in the **qosMode** parameter. The **qosMode** may contain additional information about the type of QoS report the DMIF user is interested in getting from the DMIF layer. The **response** identifies an acknowledgment (i.e., positive or negative) from the DMIF layer to a QoS monitoring request from the DMIF user.

10.4.16 DA_ChannelEvent()

DA_ChannelEvent (IN: channelHandle, mode, qosReport)

This primitive is issued by a DMIF layer to inform the DMIF user with the measured QoS for the data channel specified by the **channelHandle**. The **mode** identifies the type of QoS monitoring event (see subclause 10.5.2). The **qosReport** parameter contains the QoS profile (e.g., delay and loss parameters) measured for the channel (see subclause 10.5.3).

10.5 QoS Monitoring

10.5.1 Monitoring events

This clause introduces data structures to allow event monitoring like QoS monitoring and QoS renegotiations of established data channels in a DMIF session.

Three types of QoS monitoring events are defined: *QOS_MONITOR* or periodic monitoring depicted in section 2 of Figure 11; *QOS_VIOLATION* or notification upon violation of the requested QoS by the Application depicted in section 3 of Figure 11; and *QOS_REQUEST* or notification upon a single request from the Application depicted in section 4 of Figure 11.

The QoS monitoring event for a specific channel is initiated at any moment after the establishment of that channel (section 1 in Figure 11). The QoS monitoring process ends either when the channel is deleted or when the application explicitly (i.e., through a stop command) indicates it to the DMIF layer.

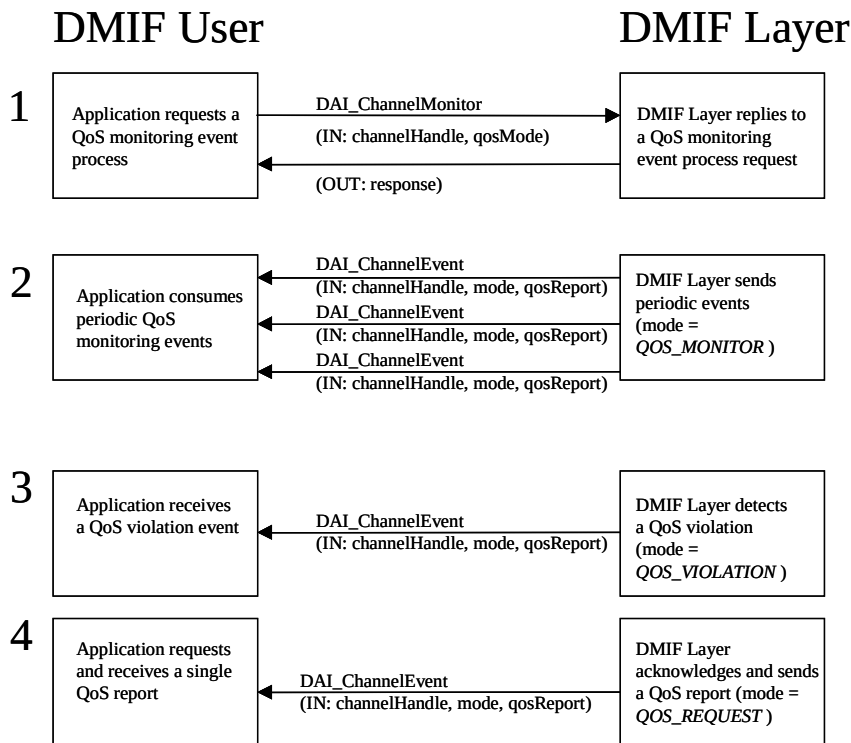


Figure 11 — QoS Monitoring Events

10.5.2 qosMode

Table 10—2 defines the format of the qosMode() structure:

Table 10—2 — qosMode() format.

Syntax	Num. Of Bytes
QosMode() {	
Mode	1
QosMonitorPeriod	2
Status	1
}	

mode: this parameter indicates the type of QoS mode. The set of modes currently defined and their semantic is summarized in Table 10—3. The application can logically combine QoS modes (e.g., QOS_MONITOR and QOS_VIOLATION). Regardless of the QoS monitoring mode, the application can always requests QoS information for a specific channel.

Table 10—3 — ISO/IEC 14496-6 (MPEG-4 DMIF) defined QoS modes.

Mode	Semantic Description
QOS_MONITOR	The DMIF layer monitors periodically the QoS of a specific data channel and reports to the application the measured QoS every qosMonitorPeriod milliseconds.
QOS_VIOLATION	The DMIF layer reports a QoS violation for a specific data channel. A QoS violation is defined when the monitored QoS does not fulfill the requested QoS profile (i.e., delay and loss parameters) specified in the qosDescriptor parameter.

qosMonitorPeriod: frequency of QoS reports by the DMIF layer to the application expressed in milliseconds.

status: this parameter indicates when a QoS monitoring process shall start or terminate. The set of status currently defined and their semantic is summarized in Table 10—4.

Table 10—4 — ISO/IEC 14496-6 (MPEG-4 DMIF) defined QoS status.

Status	Semantic Description
START	The DMIF layer starts the QoS monitoring process.
STOP	The DMIF layer terminates the QoS monitoring process.
SINGLE	The DMIF layer reports a single QoS report upon request of the application.

10.5.3 qosReport

The qosReport shall be able to carry a number of QoS metrics. The set of metrics currently defined and their semantics is summarised in Table 10—1. Table 10—5 describes the generic format of the QoS Report.

Table 10—5 — qosReport.

Syntax	Num. of Bytes
qosReport() {	
QoS_QualifierCount	1
for(l=0; i<QoS_QualifierCount; i++) {	
QoS_QualifierTag	1
QoS_QualifierDataLength	1
QoS_QualifierData	QoS_QualifierDataLength
}	
}	

QoS_QualifierCount: Provides the number of QoS qualifiers specified in the descriptor.

QoS_QualifierTag: Identifies the qualifier type.

QoS_QualifierDataLength: Is the length of the following qualifier value.

QoS_QualifierData: Provides the qualifier value.

The **QoS_QualifierTags** are shown in Table 10—1.

11 The DMIF-Network Interface

11.1 The DMIF-Network Interface key concepts

The DMIF-Network Interface (DNI) is a semantic API for the Control Plane. It abstracts the signalling between DMIF peers irrespectively of the supported delivery technologies. The parameters conveyed through the DNI are then mapped onto network dependent native signalling when possible. The parameters which cannot be mapped to native signalling are carried opaque to the native signalling in which case the necessary syntax is defined in this specification. The mappings to native signalling standards are described in clause 12.

The DMIF-Network Interface comprises the following classes of primitives:

- Session primitives, which allow the management of sessions (setup and release);
- Service primitives, which allow the management of services (attach and detach);
- Transmux primitives, which allow the management of a transmux (setup, release and config);
- Channel primitives, which allow the management of channels (add and delete).

11.2 Common syntax elements

11.2.1 DMIF Descriptors

Table 11—1 describes the generic format of a DMIF Descriptor:

Table 11—1 — General format of the DMIF Descriptor

Syntax	Num. Of Bytes
<pre>dmifDescriptor { commonDescriptorHeader() descriptorDataFields() }</pre>	

commonDescriptorHeader: Table 11—2 defines the format of the commonDescriptorHeader.

descriptorDataFields: Provides the actual descriptor data.

Table 11—2 — DMIF commonDescriptorHeader

Syntax	Num. of Bytes
<pre>commonDescriptorHeader() { DmifDescriptorType DmifDescriptorLen }</pre>	2 2

dmifDescriptorType: Defines the specific descriptor being carried. Table 11—3 defines the dmifDescriptorTypes defined in this specification.

dmifDescriptorLen: Defines the total length in bytes of the descriptorDataFields() that follow.

Table 11—3 — DMIF Descriptor Types

dmifDescriptorType	Value	Description
Reserved	0x0000	ISO/IEC 14496-6 reserved.
UuDataDescriptor	0x0001	Contains uuData as conveyed at the DAI
BypassFlexMuxDescriptor	0x0002	Indicates no flex-multiplexing is used
MPEG4SystemsFlexMuxDescriptor	0x0003	Supports MPEG-4 FlexMux as defined in ISO/IEC 14496-1:2001.
MPEG4SystemsMuxCodeDescriptor	0x0004	Supports MPEG-4 FlexMux MuxCodeTables as defined in ISO/IEC 14496-1:2001.
Reserved	0x0005-0xffff	ISO/IEC 14496-6 reserved.

11.2.1.1 uuDataDescriptor

This descriptor shown in Table 11—4 is used to carry uuData as exposed at the DAI. uuData is an opaque structure providing upper layer information; it is transparently transported through DMIF from Peer to Peer.

Table 11—4 — uuDataDescriptor

Syntax	Num. Of Bytes
UuDataBuffer	DmifDescriptorLen

uuDataBuffer: Carries the uuDataBuffer as exposed at the DAI.

11.2.1.2 BypassFlexMuxDescriptor

This descriptor shown in Table 11—5 is used to indicate that no additional multiplex tool is used on top of the particular TransMux Channel it is being associated.

Table 11—5 — BypassFlexMuxDescriptor

Syntax	Num. Of Bytes

This descriptor is empty. The dmifDescriptorLen is zero.

11.2.1.3 MPEG4SystemsFlexMuxDescriptor

This descriptor shown in Table 11—6 is used to identify a FlexMux Channel Number as defined in ISO/IEC 14496-1:2001.

Table 11—6 — MPEG4SystemsFlexMuxDescriptor

Syntax	Num. Of Bytes
FlexMuxChannelNumber	1

FlexMuxChannelNumber: Provides the number of the channel in the MPEG-4 FlexMux.

11.2.1.4 MPEG4SystemsMuxCodeDescriptor

This descriptor shown in Table 11—7 is used to carry the FlexMux MuxCodeTables as defined in ISO/IEC 14496-1:2001.

Table 11—7 — MPEG4SystemsMuxCodeDescriptor

Syntax	Num. Of Bytes
MuxCodeTable	DmifDescriptorLen

MuxCodeTable: Carries the FlexMux MuxCodeTables, whose syntax and semantics is defined in MPEG-4 Systems ISO/IEC 14496-1:2001.

11.2.2 DMIF to DMIF data

Messages which contain information to be passed between DMIF peers but that need not be inspected by the Network use the ddData structure to convey such information. This structure contains a count and a list of DMIF descriptors which are defined in subclause 11.2.1. Table 11—8 defines the format of the ddData() structure:

Table 11—8 — ddData() format

Syntax	Num. of Bytes
ddData(){ dmifDescriptorCount for(i=0;i<dmifDescriptorCount;i++) { dmifDescriptor() } }	2

dmifDescriptorCount: Indicates the total number of dmifDescriptor() structures which are included in the list.

dmifDescriptor(): Refer to subclause 11.2.1 for DMIF descriptor definitions.

11.2.3 Resource Descriptors

A Network Session consists of a relationship between two DMIF peers. Messages which are used to request resources contain a Resources() data structure. This structure contains a count and a list of resource descriptors.

Table 11—9 defines the format of the Resources() structure:

Table 11—9 — Resources() format

Syntax	Num. of Bytes
Resources(){ resourceDescriptorCount for(i=0;i<resourceDescriptorCount;i++) { ResourceDescriptor() } }	2

The **resourceDescriptorCount** field shall be set to indicate the total number of ResourceDescriptor() structures which are included in the list.

The **ResourceDescriptor()** structure shall define the type and values of a resource which is being requested or which has been assigned to a session.

Table 11—10 describes the general format of a DMIF Resource Descriptor:

Table 11—10 — General format of the Resource Descriptor

Syntax	Num. of Bytes
ResourceDescriptor { commonDescriptorHeader() resourceDescriptorDataFields() }	

The **commonDescriptorHeader** is normative and shall be included with every resource descriptor definition. Table 11—11 defines the format of the commonDescriptorHeader:

Table 11—11 — Format of the commonDescriptorHeader

Syntax	Num. of Bytes
commonDescriptorHeader() { resourceDescriptorType resourceLength resourceDataFieldCount if (resourceDescriptorType == 0xffff) { typeOwnerId typeOwnerValue } }	 2 2 2 3 3

The **resourceDescriptorType** field defines the specific resource being requested. The resourceDescriptorTypes, and their corresponding descriptors, are specified in ISO/IEC 13818-6, subclause 4.7.5. Annex E provides an extract from that subclause, where the resourceDescriptorTypes useful to this part of ISO/IEC 14496 are defined. The resourceDescriptorTypes used in DMIF are by no means limited to the ones identified in Annex E.

The **resourceLength** field defines the total length of the resourceDescriptorDataField section, which follows the commonDescriptorHeader. The value of the resourceLength field depends on the particular type of the resourceDescriptor being defined and the actual data in the resource descriptor.

The **resourceDataFieldCount** field indicates the total number of data fields in the resource descriptor, see Annex E.

The **typeOwnerId** and **typeOwnerValue** fields are defined only if the **resourceDescriptorType** field is set to 0xffff. In this situation, these fields are used to indicate that the resource descriptor data fields are defined by an organization that is outside of the scope of DSM-CC.

The **typeOwnerId** field is the first three bytes of an IEEE Organization Unique Identifier (OUI) as specified in the IEEE-802.1990.

The **typeOwnerValue** field is a **resourceDescriptorType** field defined by the owner of the **typeOwnerId** (OUI).

Annex E provides extracts from ISO/IEC 13818-6 that describe the **resourceDescriptorDataField** and the **resourceDescriptors** that are useful to this part of ISO/IEC 14496.

11.2.4 Direction parameter

Table 11—12 — Direction codes

Value	Description
0x00	ISO/IEC 14496-6 Reserved
0x01	Downstream (Sender to Receiver)
0x02	Upstream (Receiver to Sender)
0x03-0xff	ISO/IEC 14496-6 Reserved

11.2.5 Reason parameter

Table 11—13 — Reason codes

Value	Description
0x0000	Reason Normal
0x0001	Reason Error
0x0002-0x0fff	ISO/IEC 14496-6 Reserved
0x1000-0xffff	User Private

11.2.6 Response parameter

Table 11—14 — Response codes

Value	Description
0x0000	Response OK
0x0001	Response Error
0x0002-0x0fff	ISO/IEC 14496-6 Reserved
0x1000-0xffff	User Private

11.2.7 channelDescriptor and qosDescriptors

DMIF defines a ChannelDescriptor at the DMIF-Application Interface to carry all informations related to the channel configuration. This includes in particular Quality of Service parameters for both profile (e.g. loss, delay, priority, etc.) and bandwidth requirements (e.g., peak bit rate, average bit rate, etc.) for an individual Elementary Stream. The ChannelDescriptor may also include information specific to a particular application. The information carried through the DAI in the ChannelDescriptor can be further delivered across the DNI and be mapped in the DMIF Default Signaling Protocol (DDSP).

Only the semantics of the ChannelDescriptor used at the DAI is specified in subclause 10.2.1.

The exact syntax of the ChannelDescriptor and of the qosDescriptor used at the DNI and in the DMIF Default Signaling Protocol (DDSP) is specified here.

The qosDescriptor shall be able to carry a number of QoS metrics. The set of metrics currently defined and their semantic is provided in Table 11—16.

The channelDescriptor shall be able to carry a number of channel parameters. These include all the QoS metrics defined in Table 11—16; the set of parameters currently defined in addition to the QoS metrics and their semantic is provided in Table 11—18.

Table 11—15 describes the generic format of the qosDescriptor.

Table 11—15 — qosDescriptor

Syntax	Num. Of Bytes
QoS_QualifierCount	1
for(i=0; i<QoS_QualifierCount; i++) {	
QoS_QualifierTag	1
QoS_QualifierDataLength	1
QoS_QualifierData	QoS_QualifierDataLength
}	

QoS_QualifierCount: Provides the number of QoS metrics specified in the descriptor.

QoS_QualifierTag: Identifies the metric type.

QoS_QualifierDataLength: Is the length of the following metric value.

QoS_QualifierData: Provides the metric value.

The **QoS_QualifierTags** are shown in Table 11—16.

Table 11—16 — ISO/IEC 14496-6 defined QoS_QualifierTags

QoS_QualifierTag	Value	Semantic Description
Reserved	0x00	ISO/IEC 14496-6 Reserved
PRIORITY	0x01	Priority for the stream
Reserved	0x02-0x40	ISO/IEC 14496-6 Reserved
MAX_AU_SIZE	0x41	Maximum size of a PDU, as delivered over the transmux
AVG_BITRATE	0x42	Average bit rate measured over an observation time window
MAX_BITRATE	0x43	Maximum bit rate measured over an observation time window
MAX_DELAY	0x44	Maximum delay experienced by any PDU of an Elementary Stream measured over an observation time window
AVG_DELAY	0x45	Average delay experienced by the PDUs of an Elementary Stream measured over an observation time window
LOSS_PROBABILITY	0x46	Allowable probability of loss of any single PDU of an Elementary stream measured over an observation time window
JITTER_TOLERANCE	0x47	Maximum delay variation experienced by any PDU of an Elementary Stream measured over an observation time
Reserved	0x48-0x7f	ISO/IEC 14496-6 Reserved
User defined	0x80-0xff	User Private

Table 11—17 describes the generic format of the ChannelDescriptor.

Table 11—17 — ChannelDescriptor

Syntax	Num. Of Bytes
Channel_QualifierCount	1
for(l=0; i<Channel_QualifierCount; i++) {	
Channel_QualifierTag	1
Channel_QualifierDataLength	1
Channel_QualifierData	Channel_QualifierDataLength
}	

Channel_QualifierCount: Provides the number of qualifiers specified in the descriptor.

Channel_QualifierTag: Identifies the qualifier type.

Channel_QualifierDataLength: Is the length of the following qualifier value.

Channel_QualifierData: Provides the qualifier value.

The **Channel_QualifierTags** are shown in Table 11—18.

Table 11—18 — ISO/IEC 14496-6 defined Channel_QualifierTags

Channel_QualifierTag	Value	Semantic Description
Reserved	0x00-0x70	As for QoS Metrics
SYNC_GROUP	0x71	Identifies the timeline associated to a channel
SL_CONFIG_HEADER	0x72	Describes the MPEG-4 Sync Layer Packet Header format
Reserved	0x73-0x7f	ISO/IEC 14496-6 Reserved
User defined	0x80-0xff	User Private

The semantic meaning of the QoS and Channel qualifiers tags are as follows :

PRIORITY : this qualifier defines a relative measure for the priority of an elementary stream. An elementary stream with a higher priority is more important than one with a lower priority.

Table 11—19 — QoS qualifier : Priority syntax

Syntax	Num. of Bytes
Priority () { Priority }	 1

MAX_PDU_SIZE : this qualifier defines the maximum size of any PDU that can be delivered over the transmux channel.

Table 11—20 —qualifier : MaxPDUSize syntax

Syntax	Num. of Bytes
MaxPDUSize () { MaxPDUSize }	 2

AVG_BITRATE : this qualifier defines the average bitrate of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

Table 11—21 —qualifier : AvgBitrate syntax

Syntax	Num. of Bytes
AvgBitrate () {	
AvgBitrate	4
ObservationWindow	4
}	

MAX_BITRATE : this qualifier defines the maximum bitrate of the elementary stream and the time window over which it is observed. This qualifier is therefore comprised of two semantic elements.

Table 11—22 —qualifier : MaxBitrate syntax

Syntax	Num. of Bytes
MaxBitrate () {	
MaxBitrate	4
ObservationWindow	4
}	

MAX_DELAY : this qualifier defines the maximum delay expressed in microseconds experienced by any PDU of an elementary stream and the time window over which is observed expressed in milliseconds. This qualifier is therefore comprised of two semantic elements.

Table 11-23 —qualifier : MaxDelay syntax

Syntax	Num. of Bytes
MaxDelay () {	
MaxDelay	4
ObservationWindow	4
}	

AVG_DELAY : this qualifier defines the average delay expressed in microseconds experienced by the PDUs of an elementary stream and the time window over which is observed expressed in milliseconds. This qualifier is therefore comprised of two semantic elements.

Table 11-24 —qualifier : AvgDelay syntax

Syntax	Num. of Bytes
AvgDelay () {	
AvgDelay	4
ObservationWindow	4
}	

LOSS_PROBABILITY : this qualifier defines the allowable probability of loss expressed as a real number between 0 and 1 experienced by any single PDU of an elementary stream and the time window over which is observed expressed in milliseconds. This qualifier is therefore comprised of two semantic elements.

Table 11-25 —qualifier : LossProb syntax

Syntax	Num. of Bytes
LossProb () {	
LossProb	8
ObservationWindow	4
}	

JITTER_TOLERANCE : this qualifier defines the maximum delay variation expressed in microseconds experienced by any PDU of an elementary stream and the time window over which is observed expressed in milliseconds. This qualifier is therefore comprised of two semantic elements.

Table 11-26 —qualifier : JitterTol syntax

Syntax	Num. of Bytes
JitterTol () {	
JitterTol	4
ObservationWindow	4
}	

SYNC_GROUP : this channel qualifier provides an identifier for the timeline associated to a channel.

Table 11-27 — Channel qualifier : SyncGroup syntax

Syntax	Num. of Bytes
SyncGroup () {	
SyncGroup	2
}	

SL_CONFIG_HEADER : this channel qualifier describes the MPEG-4 Sync Layer Packet Header format. This qualifier is comprised by a binary sequence of bytes called slConfig, whose syntax corresponds to SLConfigDescriptor defined in ISO/IEC 14496-1:2001. The length of the slConfig is specified by slConfigLen. This qualifier is therefore comprised of two semantic elements.

Table 11-28 — Channel qualifier : SLConfigHeader syntax

Syntax	Num. of Bytes
SLConfigHeader () {	
SLConfigLen	2
SLConfig	SLConfigLen
}	

11.3 DMIF-Network Interface primitives

This subclause lists the primitives specified for the DMIF-Network Interface and describes the parameters used. The C++ like formalism used to describe the primitives only aims at capturing their semantic meaning. IN and OUT keywords allow to clearly distinguish the parameters provided to the other side of the interface from those returned from it; by no means they are meant to force a synchronous implementation. The loop() construct allows to concisely represent an array of elements.

The optional [Callback] construct indicates that there are always pairs of primitives (with and without the suffix Callback) with identical parameters. A primitive without the 'Callback' is issued by the DMIF peer initiating the primitive, as a consequence the corresponding primitive with the 'Callback' is issued at the target DMIF peer (see Walkthrough in Annex B).

The primitives are:

- DN_SessionSetup[Callback] (IN: networkSessionId, calledAddress, callingAddress, compatibilityDescriptorIn; OUT: response, compatibilityDescriptorOut)
- DN_SessionRelease[Callback] (IN: networkSessionId, reason; OUT: response)
- DN_ServiceAttach[Callback] (IN: networkSessionId, serviceId, serviceName, ddDataIn(); OUT: response, ddDataOut())
- DN_ServiceDetach[Callback] (IN: networkSessionId, serviceId, reason; OUT: response)
- DN_TransMuxSetup[Callback] (IN: networkSessionId, loop(TAT, qosDescriptor; resources()); OUT: loop(response, resources()))
- DN_TransMuxRelease[Callback] (IN: networkSessionId, loop(TAT); OUT: loop(response))
- DN_ChannelAdd[Callback] (IN: networkSessionId, serviceId, loop(CAT, channelDescriptor, direction, ddDataIn()); OUT: loop(response, TAT, ddDataOut()))
- DN_ChannelAdded[Callback] (IN: networkSessionId, serviceId, loop(CAT, channelDescriptor, direction, TAT, ddDataIn()); OUT: loop(response, ddDataOut()))
- DN_ChannelDelete[Callback] (IN: networkSessionId, loop(CAT, reason); OUT: loop(response))
- DN_TransMuxConfig[Callback] (IN: networkSessionId, loop(TAT, ddDataIn()); OUT: loop(response))

- DN_UserCommand[Callback] (IN: networkSessionId, ddDataIn(), loop(CAT))
- DN_UserCommandAck[Callback] (IN: networkSessionId, ddDataIn(), loop(CAT); OUT: response, ddDataOut())

The parameters have the following semantics:

calledAddress: the network address of the Target DMIF peer. The Originating DMIF peer, issuing DN_SessionSetup, strips this addressing portion of a URL (see DAI definitions) to identify the location of the Target DMIF where a requested service resides. Depending on the form of the URL the originating DMIF peer may have to invoke directory service to obtain the calledAddress.

callingAddress: the network address of the Originating DMIF peer, i.e. the peer issuing DN_SessionSetup.

CAT: (Channel Association Tag) is an Association Tag which uniquely identifies a channel end-to-end within a Network Session (identified by networkSessionId) and is not changed during its lifetime within this Network Session. It is generated locally at a DMIF peer prior to issuing a DN_DownstreamChannelAdd() or DN_UpstreamChannelAdd().

channelDescriptor: see DAI definitions in subclause 10.3. See also subclause 11.2.4.

compatibilityDescriptor: in the case when native signalling does not provide capability exchange, this parameter instead (opaque to native signalling) provides a descriptor to enable capability exchange between DMIF peers as syntactically defined in subclause 13.1.

ddDataIn(): is an opaque structure providing Delivery layer information; it is transparently transported through native signalling from the local Peer to the remote Peer. The generic syntax of such data is provided in subclause 11.2.2.

ddDataOut(): is an opaque structure providing Delivery layer information; it is transparently transported through native signalling from the remote Peer to the local Peer. The generic syntax of such data is provided in subclause 11.2.2.

direction: see DAI definitions in subclause 10.3. See also subclause 11.2.4.

networkSessionId: uniquely identifies a Network Session end-to-end. It is generated locally at the originating DMIF peer prior to issuing DN_SessionSetup.

qosDescriptor: is a parameter containing the complete description of the Quality of Service requested for a particular channel or transmux. Its syntax and semantic definitions are given in subclause 11.2.7.

reason: see DAI definitions in subclause 10.3. See also subclause 11.2.5.

resources(): is a structure containing a count and a list of DSM-CC resource descriptors which are defined in subclause 4.7 of ISO/IEC 13818-6. See also subclause 11.2.3.

response: see DAI definitions in subclause 10.3. See also subclause 11.2.6.

serviceld: uniquely identifies a service end-to-end within a Network Session (identified by networkSessionId). It is generated locally at a DMIF peer prior to issuing DN_ServiceAttach. It corresponds to a serviceName which is opaque to DMIF.

serviceName: see DAI definitions in subclause 10.3.

TAT: (Transmux Channel Association Tag) is an Association Tag which uniquely identifies a Transmux Channel end-to-end within a Network Session (identified by networkSessionId) and is not changed during its lifetime within this Network Session. It is generated locally at a DMIF peer prior to issuing a DN_TransMuxSetup.