ETSI TS 103 205 V1.1.1 (2014-03)



Digital Video Broadcasting (DVB); Extensions to the CI Plus™ Specification







Reference DTS/JTC-DVB-333 Keywords CI Plus, DVB

ETSI

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

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

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

Important notice

The present document can be downloaded from: http://www.etsi.org

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.

Information on the current status of this and other ETSI documents is available at

http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services: <u>http://portal.etsi.org/chaircor/ETSI_support.asp</u>

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2014.
© European Broadcasting Union 2014.
All rights reserved.

DECTTM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members. **3GPP**TM and **LTE**TM are Trade Marks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

GSM® and the GSM logo are Trade Marks registered and owned by the GSM Association.

Contents

Intell	lectual Property Rights	9
Forev	word	9
Intro	duction	9
1	Scope	10
2	References	10
2.1	Normative references	
2.1	Informative references	
3	Definitions and abbreviations	11
3.1	Definitions	11
3.2	Abbreviations	13
4	CI Plus TM extensions overview	15
4.1	Introduction	
4.2	Multi-stream reception	
4.3	IP-delivered content	
4.3.1	General	
4.3.2	IP delivery modes	18
4.3.2.	.1 Host player mode	18
4.3.2.	I ·/	
4.3.3	•	
4.3.3.		
4.3.3.	\mathcal{S} (1)	
4.3.3.	\mathcal{S} \mathcal{A}	
4.3.3.		19
4.4	CI Plus TM browser extensions	
4.5 4.6	CICAM application launching	
4.0 4.7	Usage Rules Information extensions	
4.8	Watermarking and transcoding	
5	General requirements	
5 5.1	Backwards compatibility	
5.2	Watermarking and transcoding	
5.2	PES level scrambling	
6	Multi-stream reception	
6.1 6.2	General	
6.2.1		
6.2.2	Local TS identifier	
6.2.3	Multiplexed TS packet order, delay and delay variation	
6.2.4	Scrambling cipher and CCK usage	
6.2.5	Host Service Shunning	
6.2.6	TS clock	
6.2.7	Multi-stream operation with multiple CICAMs	
6.3	PID Selection	
6.3.1	General	25
6.3.2	Default PID selection	
6.3.3	Default PID selection for frequency tune	
6.3.4	PID selection priority	
6.3.5	CICAM initiated update	
6.3.6	Change in ES selection	
6.3.7	Host initiated PID addition or removal	
6.3.8	Selected services from the same broadcast TS	
6.3.9	Example PID selection sequence	27

Resources for multi-stream operation.	
General	29
Multi-stream resource	29
General	29
CICAM multi-stream capability APDU	29
PID select request APDU	30
PID select reply APDU	
Content Control resource	32
General	32
Content Control APDU extensions	32
cc PIN reply APDU	32
* *	
* *	
•	
•	
1	
_	
· •	
• • • • •	
delivery Host player mode	46
General	46
TS Interface modes	47
Command Interface	47
General	
Playback initiation	47
Playback execution	
Playback termination	
Sample decryption resource	
Resource usage	
sd_info_req APDU	
sd_info_reply APDU	
· ·	
sd start reply APDU	
	General. Multi-stream resource General. CICAM multi-stream capability APDU. PID select request APDU. PID select replex APDU. Content Control resource General. Content Control APDU extensions cc_PIN_reply APDU cc_PIN_event APDU. Content Control protocol extensions. Clay It transmission and acknowledgement protocol Record Start protocol. Record Stop protocol extension. Change Operating Mode protocol extensions. CICAM to Host License Exchange protocol extensions. CICAM to Host License Exchange protocol extensions. Conditional Access Support resource General. ca_pmt APDU. ca_pmt_reply APDU Multi-stream Host Control resource. General. tune_broadcast_req APDU tune_triplet_req APDU tune_triplet_req APDU tune_triplet_req APDU tune_triplet_req APDU tune_triplet_req APDU tune_reply APDU Application MMI resource. General. RequestStart APDU RequestStart APDU FileAck APDU FileRequest APDU AppAbortRequest APDU AppAbortRequest APDU Migh_Level MMI resource General. enq APDU answ APDU. menu_answ APDU. list APDU display_control APDU display_reply APDU delivery Host player mode General. Playback termination Sample decryption resource Resource usage sd_info_req APDU sd_info_req APDU

7.4.6	sd_update APDU	
7.4.7	sd_update_reply APDU	58
7.5	TS Interface	58
7.5.1	General	58
7.5.2	TS content carriage	58
7.5.2.1	Host output	58
7.5.2.2	CICAM output	60
7.5.2.3	Multiple TS Sample Tracks	61
7.5.3	Non-TS content carriage	61
7.5.3.1	Host output	61
7.5.3.1.1	General	61
7.5.3.1.2	Transmitting Samples	61
7.5.3.1.3	Managing multiple Tracks	63
7.5.3.1.4	Track list update	65
7.5.3.2	TS packet headers	
7.5.3.3	CICAM output	
7.5.3.4	ISOBMFF Samples	
7.5.3.4.1	SST Section	71
7.5.3.4.2	ISOBMFF Sample packetization	71
7.5.4	CICAM Buffering	
7.5.4.1	General	
7.5.4.2	Buffer size indication	72
7.5.4.3	Buffer level indication.	72
7.5.4.4	Flushing CICAM buffers	
7.5.5	comms table	72
7.5.5.1	General	
7.5.5.2	Mapping of comms sections into TS packets	
7.5.5.3	Mapping of comms sections into TS packet Adaptation Fields	
7.5.5.4	Coding of table_id values	
7.5.5.5	Sample Start Table (SST)	
7.5.5.6	Sample End Table (SET)	
7.5.5.7	Flush Table (FLT)	
7.5.5.8	Buffer Level Table (BLT)	
7.5.5.9	Descriptors	
7.5.5.9.1	•	
7.5.5.9.2	TO 6	
7.5.5.9.3		78
7.5.6	URI	
		=-
	P delivery CICAM player mode	
8.1	General	
8.2	Player controls	
8.3	Session initialisation	
8.3.1	General	
8.3.2	Host-initiated playback	
8.3.3	CICAM-initiated playback	
8.4	Communication errors	
8.5	Trick mode support	
8.6	Session termination	
8.6.1	General	
8.6.2	Unrecoverable error	
8.6.3	Termination by the user	
8.6.4	End of content	
8.7	CICAM player mode sequence	
8.8	CICAM Player resource	
8.8.1	General	
8.8.2	CICAM Player resource APDUs	
8.8.3	CICAM_player_verify_req APDU	
8.8.4	CICAM_player_verify_reply APDU	
8.8.5	CICAM_player capabilities_req APDU	
8.8.6	CICAM_player_capabilities_reply APDU	
8.8.7	CICAM player start req APDU	87

8.8.8	CICAM_player_start_reply APDU	88
8.8.9	CICAM_player_play_req APDU	
8.8.10	CICAM_player_status_error APDU	89
8.8.11	CICAM_player_control_req APDU	
8.8.12	CICAM_player_info_req APDU	91
8.8.13	CICAM_player_info_reply APDU	92
8.8.14	CICAM_player_stop APDU	92
8.8.15	CICAM_player_end APDU	93
8.8.16	CICAM_player_asset_end APDU	
8.8.17	CICAM_player_update_req APDU	
8.8.18	CICAM_player_update_reply APDU	
8.8.19	CICAM player resource summary	
	* *	
9 C	CICAM file retrieval	95
9.1	General	
9.2	File system offer APDU	96
9.3	File System Ack APDU	
9.4	File request APDU	98
9.5	File acknowledge APDU	98
9.6	Auxiliary file system resource summary	98
9.7	Auxiliary file system and Application MMI resource coordination	98
10 Y		
	ow Speed Communication resource version 4	
10.1	General	
10.2	Host IP configuration information	
10.3	Information about IP streams	99
10.4	IP multicast	100
10.5	IP source ports	100
10.6	Host to CICAM delivery	100
10.7	CICAM to Host delivery	100
10.8	IP flow control	100
10.9	comms_info	101
10.9.1	General	101
10.9.2	comms_info_req APDU	101
10.9.3	comms_info_reply APDU	
10.10	comms_IPconfig	
10.10.1	General	
10.10.2	comms_IP_config_req	
10.10.3	comms_IP_config_reply	
10.11	Comms Cmd modification	
10.11.1	General	
10.11.2	Comms Cmd hybrid descriptor	
10.11.2	Comms Cmd multicast descriptor	
10.11.5	Low Speed Communications resource types modification	
10.12.1	General	
10.12.1	CICAM Flow Control	
10.12.2	Disconnection behaviour	
10.12.3	Data transfer across the TS Interface	
10.12.4 10.12.4.1		
	1 ,	
10.12.4.2	2 Adaptation field usage	108
11 U	Jsage Rules Information version 3	109
12 C	CICAM applications	
12.1	General	
12.2	CI Plus TM Browser extensions	111
12.2.1	InteractionChannelExtension	
12.2.2	ICStreamingExtension	
12.2.3	ICEncryptedStreamExtension	
12.2.4	Video scaling	
12.2.4.1	General	
12.2.4.2	Set of features.	
12.2.4.3	GetEngineSupport	
r.J		1 1 1

12.3	Application life cycle	112			
12.4	Application Coordination Framework				
12.4.1	General	113			
12.4.2	Modifications to referenced specifications	114			
12.4.3	CICAM application launch mechanisms				
12.4.3.1					
12.4.3.2					
12.4.3.3					
12.4.3.3					
12.4.3.3					
12.4.3.3					
12.4.4	Broadcast and CICAM application coordination				
12.4.4.1	**				
12.4.4.2					
12.4.4.3					
12.4.4.4					
12.4.4.5					
12.4.4.6					
12.4.5	Application coordination scenarios.				
12.4.5.1	11				
12.4.5.2					
12.4.5.3					
12.4.5.4					
12.5	Host application environments				
12.5.1	General				
12.5.2	Application provision on a CICAM	124			
12.5.3	Determining Host application environment support				
12 F	NAD II A COLA 1 1 2	105			
	OVB Host Control version 3				
13.1	General				
13.2	DVB Host Control version 3 APDUs.				
13.2.1	tune_broadcast_req APDU				
13.2.2	tune_triplet_req APDU				
13.2.3	tune_lcn_req APDU				
13.2.4	tune_ip_req APDU				
13.2.5	tuner_status_req APDU				
13.2.6	tuner_status_reply APDU	128			
14 C	CICAM Virtual Channel	129			
14.1	Introduction	129			
14.2	Application Information version 4	130			
14.2.1	General	130			
14.2.2	enter_cicam_channel APDU	130			
15 (Drawston Drafile version 2	120			
	Operator Profile version 2				
15.1	Introduction				
15.2	Profile Type 2				
15.2.1	Introduction.				
15.2.2	Initialisation and profile discovery				
15.2.3	Moving between profiles				
15.2.4	LCN collision management				
15.3	CICAM Virtual Channel				
15.3.1	CICAM Virtual Channel descriptor				
15.3.2	Selection of the CICAM Virtual Channel				
15.3.3	Operator NIT management APDU				
15.4	IP delivered services.				
15.4.1	operator_status() APDU				
15.4.2	IP service discovery and update				
15.4.3	operator_osdt_request APDU				
15.4.4	operator_osdt_reply APDU	136			
16 F	High-Level MMI	136			

Resources summary	137
Credential Specification: Parameters exchanged in APDUs	142
Descriptor identification and location	144
Schemas	145
	145
ttribute groups	
	145
Type	
(informative)	148
Management of a DiSEqC TM switch under CICAM control	150
CICAM application signalling	153
	153
oadcast applications	153
oadcast and CICAM broadcast applications	153
	154
	Sype

Intellectual Property Rights

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

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

Foreword

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

NOTE:

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

European Broadcasting Union CH-1218 GRAND SACONNEX (Geneva) Switzerland

Tel: +41 22 717 21 11 Fax: +41 22 717 24 81

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

Introduction

The DVB Common Interface specifications EN 50221 [1] and TS 101 699 [2] describe a system whereby a removable Conditional Access CICAM, given the appropriate rights, unscrambles protected content and routes it back to the Host over the same interface. The Common Interface connector is an industry standard PCMCIA slot. The DVB Common Interface specifications were extended by the CI PlusTM specification [3], developed by CI PlusTM LLP, which provides common methods (i.e. methods that are independent of the up-stream CA system) for mutual authentication of the CICAM and Host, and link encryption over the return interface from the CICAM to the Host.

1 Scope

The present document specifies extensions to the CI PlusTM V1.3 specification [3], which was produced and continues to be published by CI PlusTM LLP.

2 References

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

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

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

2.1 Normative references

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

[1]	CENELEC EN 50221 (02-1997): "Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications".
[2]	ETSITS 101 699 (V1.1.1) (11-1999): "Digital Video Broadcasting (DVB); Extensions to the Common Interface Specification".
[3]	CI Plus TM specification (V1.3.1) (09-2011): "Content Security Extensions to the Common Interface".
NOTE:	Available from: http://www.CI Plus.com/data/CI Plus specification V1.3.1.pdf .
[4]	Recommendation ITU-T H.222.0 (2006)/ISO/IEC 13818-1:2007: "Information technology Generic coding of moving pictures and associated audio information: Systems".
[5]	IETF RFC 4122: "A Universally Unique IDentifier (UUID) URN Namespace".
NOTE:	Available from: http://tools.ietf.org/html/rfc4122.
[6]	ETSI ES 202 184 (V2.3.1) (12-2012): "MHEG-5 Broadcast Profile".
[7]	ETSITS 102 809 (V1.1.1): "Digital Video Broadcasting (DVB); Signalling and carriage of interactive applications and services in Hybrid broadcast/broadband environments".".
[8]	ISO/IEC 14496-12:2012: "Information technology Coding of audio-visual objects Part 12: ISO base media file format".
[9]	ISO/IEC 14496-14:2003: "Information technology Coding of audio-visual objects Part 14: MP4 file format".
[10]	ETSI EN 300 468 (V1.13.1) (08-2012): "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".
[11]	ETSI TS 101 162: "Digital Video Broadcasting (DVB); Allocation of identifiers and codes for Digital Video Broadcasting (DVB) systems".
[12]	Open IPTV Forum: "Release 1 Specification, Volume 5 - Declarative Application Environment", V1.2, August 2012.

NOTE: Available from: http://www.oipf.tv/specifications.

[13]	Open IPTV Forum: "Release 1 Specification, Volume 3 - Content Metadata", V1.2, August 2012.
NOTE:	Available from: http://www.oipf.tv/specifications.
[14]	ISO/IEC 23001-7:2012: "Information technology - MPEG systems technologies - Part 7: Common encryption in ISO base media file format files".
[15]	ISO/IEC 23009-1: "Information technology Dynamic adaptive streaming over HTTP (DASH) Part 1: Media presentation description and segment formats".
[16]	ETSI TS 102 034 (V1.4.1) (08-2009): "Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks".
[17]	IETF RFC 768: "User Datagram Protocol".
[18]	IETF RFC 791: "Internet Protocol".
[19]	IETF RFC 793: "Transmission Control Protocol".
[20]	IETF RFC 3376: "Internet Group Management Protocol, Version 3".
[21]	IETF RFC 1112: "Host extensions for IP multicasting".
[22]	IETF RFC 2460: "Internet Protocol, Version 6 (IPv6) Specification".
[23]	IETF RFC 4443: "Internet Control Message Protocol (ICMPv6) for the Internet Protocol, Version 6 (IPv6) Specification".
[24]	High-bandwidth Digital Content Protection System, Interface Independent Adaptation, Revision 2.2.
NOTE:	Available from: http://www.digital-cp.com/hdcp_specifications .
[25]	Digital Transmission Content Protection Specification Volume 1 (Informational Version) Revision 1.7.
NOTE:	Available from: http://www.dtcp.com/specifications.aspx.
[26]	ETSI TS 102 809 (V1.2.1) (07-2013): "Digital Video Broadcasting (DVB); Signalling and carriage of interactive applications and services in Hybrid broadcast/broadband environments".

2.2 Informative references

The following referenced document is not necessary for the application of the present document but it assists the user with regard to a particular subject area.

[i.1] ETSI TS 102 727 (01-2010): "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.2.2".

3 Definitions and abbreviations

3.1 Definitions

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

 $\textbf{application coordination framework:} \ \text{set of CI Plus}^{\text{TM}} \ \text{specific rules around the coordination of broadcast and CICAM applications}$

NOTE: As specified in clause 12.4 of the present document.

application MMI resource: As defined in clause 14.4 of CI PlusTM 1.3 [3].

NOTE: CI PlusTM V1.3 [3] also uses this term to mean the CI PlusTM browser. TS 101 699 [2] also uses this term to mean an application started using the application MMI resource or to mean an application controlled by that resource.

background tune: tune operation in multi-stream mode whereby the received service is not for presentation to the user at the time of reception

broadcast application: broadcast signalled application or an application started by a broadcast application even if itself not signalled in a DVB service

broadcast signalled application: application signalled in a DVB service which may be carried in-band in that service (e.g. with DSM-CC object carousel) or out-of-band (e.g. via HTTP or on the CICAM auxiliary file system)

CICAM application: application provided by the CICAM, either using its auxiliary file system or using the application MMI resource

CICAM broadcast application: broadcast application loaded from the CICAM auxiliary file system

CICAM AppMMI application: application launched by the CICAM using the CI PlusTM Application MMI resource

DVB service: service signalled or announced in a way that is defined by DVB specifications, including DVB-SI [10], SD&S [16] and OSDT (see clause 15 and annex C)

foreground tune: tune operation in multi-stream mode whereby the received service is for presentation to the user

Host: IRD that includes a CI PlusTM compliant CICAM slot

Input Mode: mode of operation of the TS Interface whereby the CICAM generates a new TS that is provided for the Host

IP-delivered content: AV content that is received via the IP network interface of an IRD

IP delivery CICAM player mode: mode of handling IP-delivered content whereby the CICAM handles the delivery protocols and encapsulation format of the content

IP delivery Host player mode: mode of handling IP-delivered content whereby the Host handles the delivery protocols and encapsulation format of the content

IP network interface: wired or wireless IRD interface that supports IP based communications

Local TS: sequence of TS packets in which each TS packet has the same Local TS Identifier

Local TS Identifier: unique number allocated to a Local TS, whereby this number replaces the sync byte in each TS packet header

Multi-stream Mode: mode of operation that allows multiple Local TSs to be carried over the TS Interface

Normal Mode: mode of operation of the Local TS for the carriage of a conventional TS

Sample: logical segment of data

NOTE: For encrypted content a Sample corresponds to a segment of data encrypted with one set of encryption parameters.

Sample Mode: mode of operation of the Local TS for the carriage of Samples

Single-stream Mode: TS Interface mode of operation that is compliant with the DVB-CI specification [1]

Track: sequence of related Samples in a content file in IP delivery Host player mode

Trust Authority: entity that governs compliance and robustness for CICAM and Host implementations according to the present document

Tuner: functionality of an IRD that can deliver a TS containing one or more DVB services

Virtual Channel: Host channel list item provided by the CICAM to enable the launch of a CICAM menu from within the Host channel line-up

3.2 Abbreviations

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

ADQ Application Domain Query
AIT Application Information Table
AKH Authentication Key Host

AKM Authentication Key Module (CICAM)
APDU Application Protocol Data Unit
API Application Programming Interface
ASN.1 Abstract Syntax Notation One

AV Audio-Video

BCG Broadband Content Guide
BER Basic Encoding Rules
BLT Buffer Level Table
bslbf bit string, left bit first
CA Conditional Access

CA_PMT Conditional Access Program Map Table

CAS CA System

CASD Content Access Streaming Descriptor

CAT Conditional Access Table

CC Content Control
CCK Content Control Key
CENC Common Encryption
CI Common Interface
CICAM CI CA Module
CIS CI SendMessage

CIV Content Initialisation Vector

DASH Dynamic Adaptive Streaming over HTTP

DH Diffie Hellman

DHCP Dynamic Host Configuration Protocol

DHPH DH Public key Host

DHPM DH Public Key CICAM Module

DNS Domain Name Service
DOT Digital Only Token
DRM Digital Rights Management
DSD Delivery System Descriptor

DTCP Digital Transmission Content Protection

Entitlement Control Message

DTV Digital Television

DVB Digital Video Broadcasting

DVB-C DVB Cable
DVB-S DVB Satellite
DVB-T DVB Terrestrial

ECM

EIT Event Information Table

EMI Encryption Mode Indicator

EMM Entitlement Management Message

EPG Electronic Programme Guide

ES MPEG-2 Elementary Stream

FEC Forward Error Correction

FLT Flush Table GBR Great Britain

HbbTV[®] Hybrid Broadcast Broadband Television

HD High Definition

HDCP High-bandwidth Digital Content Protection system

HDMI[®] High Definition Multimedia Interface

HTTP HyperText Transfer Protocol ICT Image Constraint Token

IGMP Internet Group Management Protocol

IP Internet Protocol IPTV IP Television

IRD Integrated Receiver Decoder
ISO International Standards Organisation

ISOBMFF ISO Base Media File Format

IV Initialisation Vector KID Key Identifier

LCN Logical Channel Number LLP Limited Liability Partnership

LNB Low Noise Block

LSC Low Speed Communications

LTS Local TS

MAC Media Access Control

MAS(N) Multiple Audio Streams (Number)

MHEG Multimedia and Hypermedia Experts Group

MHP® Multimedia Home Platform
MMI Man-Machine Interface
MPD Media Presentation Description
MPEG Motion Picture Experts Group

MSB Most Significant Bit

MVS(N) Multiple Video Streams (Number)
NAT Network Address Translation
NIT Network Information Table

OIPF Open IPTV Forum OSDT Online SDT

PAT Program Association Table

PCMCIA PC Memory Card International Association

PES Packetised Elementary Stream

PID Packet IDentifier

PIN Personal Identification Number
PKI Public Key Infrastructure
PMT Program Map Table

Program Specific Information PSI **PVR** Personal Video Recorder **RCT** Redistribution Control Token **ROT** Root of Trust (i.e. Trust Authority) **RTP** Real Time Transport Protocol **RTSP** Real Time Streaming Protocol SAC Secure Authenticated Channel Specific Application Support SAS

SD Standard Definition
SDT Service Description Table
SET Sample End Table
SI Service Information

SPTS Single Programme Transport Stream SRM System Renewability Message

SSM Set Subtitle Mode SST Sample Start Table

TCP Transmission Control Protocol
TS MPEG-2 Transport Stream
TSC Transport Scrambling Control
UDP User Datagram Protocol

UI User Interface

uimsbf unsigned integer most significant bit first

URI Usage Rules Information
URL Uniform Resource Locator
URN Uniform Resource Name
UTF Unicode Transformation Format
UUID Universally Unique IDentifier

VoD Video on Demand

XML Extensible Markup Language

4 CI Plus[™] extensions overview

4.1 Introduction

The present document provides various extensions to the CI PlusTM V1.3 specification [3], which maintains its validity for CICAM and Host implementations according to the present document, unless specific clauses are amended or excluded explicitly in the present document.

The following is a list of the features specified as extensions to CI PlusTM V1.3 [3]. Each of these aspects is described informatively in the following clauses and their normative specification is contained in subsequent clauses of the present document:

- Multi-stream handling.
- IP-delivered content.
- CI PlusTM browser extensions.
- CICAM application launching.
- URI (usage rules information) extensions.
- Watermarking and transcoding capability.

Figure 1 provides a CI PlusTM system overview that is extended from that provided in Figure 4.1 in CI PlusTM V1.3 [3], to include the new stream handling features specified in the present document.

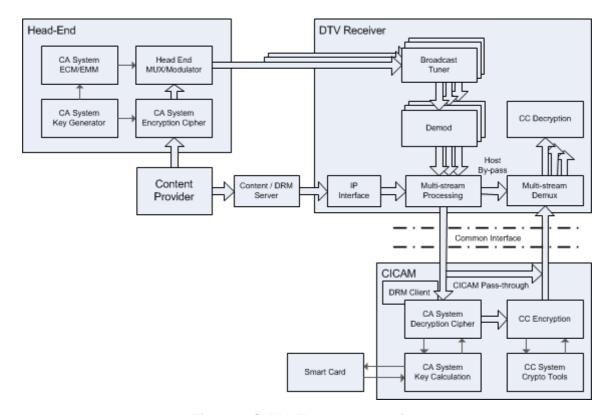


Figure 1: CI Plus™ system overview

The DTV Receiver (Host) can route broadcast services received from more than one broadcast tuner to the CICAM via the multi-stream processing function. Multi-stream processing consists of the filtering of incoming broadcast TSs to isolate the component streams actually needed by the CICAM, and the multiplexing of streams from multiple tuners over the TS Interface. Multi-stream handling is described in more detail in clause 4.2 and specified in clause 6.

The Host can route content received from an IP network interface to the CICAM also via the multi-stream processing function, or directly if the Host supports the routing of only one service to the CICAM at a time. IP-delivered content is formatted for carriage over the TS Interface as specified in the present document. Clause 4.3 contains a full introduction to IP delivery and the feature is specified in clauses 7 and 8.

Figure 2 shows the end-to-end scope of protection schemes as an extended version of that provided in figure 5.1 in CI PlusTM V1.3 [3], to include the new feature of IP delivery, where content is nominally protected by a DRM system.

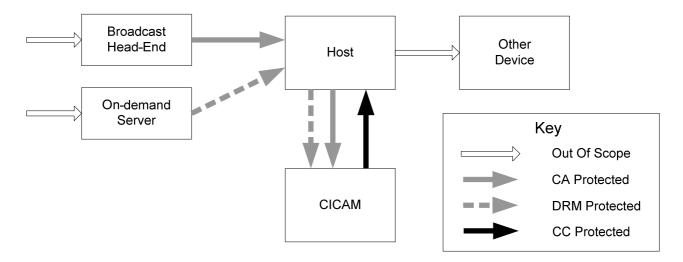


Figure 2: Scope of protection schemes

The present document does not contain any extensions to the following features specified in CI PlusTM V1.3 [3]:

- authentication mechanisms (specified in clause 6 of [3]);
- secure authenticated channel (specified in clause 7 of [3]);
- PKI and certificate details (specified in clause 9 of [3]);
- Host service shunning (specified in clause 10 of [3]), except for a provision with multi-stream and IP delivery handling;
- CI PlusTM man machine interface resource (specified in clause 13 of [3]).

The present document extends Annex H of the CI PlusTM V1.3 specification [3] with a new datatype_id, and extends Annex L of the same document with the new resources and APDUs defined in the present document. No further annexes of the CI PlusTM V1.3 specification [3] are amended in the present document. In particular, no extensions or changes are made to the CI PlusTM electrical specification, contained in Annex K of the CI PlusTM V1.3 specification [3].

Clause 5 of the present document contains general requirements for Host and CICAM implementations according to the present document.

Clauses 6 and onwards contain the specification of the features of CI PlusTM that are introduced with the present document. These consist of completely new features as well as amendments and restrictions against CI PlusTM V1.3 [3].

Annex E provides an informative account of an automatic installation procedure of the user's satellite reception equipment under the control of an operator-specific CICAM for the most common $DiSEqC^{TM}$ installations.

4.2 Multi-stream reception

The CI PlusTM V1.3 specification [3] is based on an architecture that supports the routing of only a single TS via the TS Interface, through the CICAM and back to the Host, and the decryption of one or more services in that TS. This imposes a limitation on the flexibility of implementations to support, for example, the concurrent viewing and recording of services that are delivered on different physical channels, whereby both services are protected by the CA system implemented in the CICAM. This example reflects device functionality that is already commonly implemented in PVR devices, but currently two discrete instances of a CICAM would be needed, one for each physical tuner equipped in the PVR. This model is, however, generally not economically viable.

The present document defines extensions to CI PlusTM that enable the carriage of multiple services from multiple broadcast tuners and/or the IP network interface on the Host over the TS Interface with a single attached CICAM. CAS/DRM protection may be applied to multiple services while access is provided via a single CICAM. The scope diagram in figure 1 depicts the example of the Host containing three broadcast tuners for multi-stream handling. Multi-stream handling requires that the TS Interface be put into a different mode of operation, called multi-stream mode, compared to the conventional TS Interface according to the DVB-CI specification [1], where only a single TS may be fed to the CICAM for the decryption of a service contained in that TS.

The new features related to multi-stream handling are specified in clause 6.

4.3 IP-delivered content

4.3.1 General

Many broadcast service providers and network operators already deliver content over IP, often as a supplement to broadcast services. Host devices are increasingly capable of being connected to broadband IP networks, and have hybrid broadcast broadband capability. Previous versions of the CI specifications do not include the facility to route IP-delivered content over the TS Interface, other than a conventional CA-protected TS, hence user access to IP-delivered content that is DRM protected needs a separate DRM client in the Host. The present document contains extensions for the routing and decryption of protected content delivered over IP to a network interface on the Host using a DRM client in the CICAM. This enables service providers and network operators to provide a single CICAM that can receive and decrypt both broadcast and IP-delivered content offerings with a single CICAM.

The container formats and delivery protocols above IP used for the delivery of content between a server and the Host are out of scope of the present document, except to the extent that they influence the behaviour of the content transfer across the CI. The provision of IP-delivered protected content by the Host to the CICAM is however designed to be suitable for all of the widely adopted formats at the time the present document is published.

The present document specifies support for both TS and ISOBMFF-encapsulated IP-delivered content. TS format IP-delivered content does not require any re-encapsulation for carriage over the TS Interface. ISOBMFF [8] or MP4FF [9] container IP-delivered content is encapsulated into a CI PlusTM IP-delivery specific TS-based container format that is applied for the transfer between Host and CICAM, and back to the Host. When carrying non-TS container content, the TS Interface is said to be operating in Sample Mode, otherwise when carrying a conventional TS it is in Normal Mode.

In Sample Mode, DRM metadata and control information related to the IP-delivered content item being received is passed from the Host to the CICAM via the Command Interface before playback can be started. DRM metadata that changes between Samples is carried with the Sample data on the TS Interface.

In Sample Mode the CICAM will usually have to buffer incoming data from the Host until it is ready to decrypt Samples. For this reason the generally applicable rule with conventional TS of constant packet delay through the CICAM does not apply in Sample Mode.

IP delivery in Sample Mode is applicable to both single-stream and multi-stream operation, as described in clause 4.2, with Hosts and CICAMs that are compliant with the present document.

Two IP delivery modes are foreseen in the present document, differentiated by the level of cognisance of the Host with the IP delivery protocols and formats. They are introduced in clause 4.3.2.

Some illustrative examples of IP delivery use cases are provided in clause 4.3.3.

4.3.2 IP delivery modes

4.3.2.1 Host player mode

This mode covers a topology in which the Host is connected to a broadband network, and handles the protocols by which the content is delivered. The Host also processes the encapsulation, or container format, for the content. The Host passes the content to the CICAM for DRM decryption. The CICAM returns the decrypted content, which it may re-encrypt using the CI PlusTM Content Control system already defined in the CI PlusTM V1.3 specification [3], to the Host. The content encapsulation and transfer protocol used across the Host to CICAM interface and vice versa are specified in clause 7.4.

This approach is analogous to the operation of Hosts and CICAMs for broadcast content, in which the CICAM provides the proprietary protection client to decrypt the content and may deliver the decrypted content securely back to the Host using the CI PlusTM Content Control system.

IP delivery Host player mode is specified in clause 7.

4.3.2.2 CICAM player mode

This mode covers a topology in which the Host is connected to a broadband network, but does not handle the protocols above UDP or TCP by which the content is delivered, or its encapsulation formats or codecs, but instead passes the IP packet payloads to the CICAM. The CICAM interprets the encapsulation and manages the delivery protocols, and translates them as necessary into a format supported by the Host. The content is returned from the CICAM to the Host as a SPTS that the Host is able to consume directly.

IP delivery CICAM player mode can allow a service to use future standards for delivery and content encapsulation that are unknown to the Host, if these are supported by the CICAM. It also allows for proprietary methods of delivery and content encapsulation (above TCP or UDP) to be used, provided that the CICAM can translate these protocols/formats into the interface format defined for reception of the content from the CICAM by the Host.

IP delivery CICAM player mode is specified in clause 8.

4.3.3 IP delivery use cases

4.3.3.1 General

The delivery of content over IP can use a number of approaches. The principal methods are illustrated in the following clauses by three IP delivery use cases. This set of use cases is not intended to be an exhaustive representation of all possibilities.

4.3.3.2 Linear and VoD Streaming (pull)

The content is streamed from the server using a pull protocol such as HTTP. The Host might be able to perform trick play, for example to skip forwards or backwards within a content item. The Host is also able to provide fast forward and reverse modes by either requesting the content faster than real time or by requesting chunks of the content (e.g. I-frames).

This use case also includes adaptive streaming of the content, whereby the server provides different versions of the same content, coded at different bit-rates. Adaptively streamed content is accompanied by a manifest containing initialisation data for the content.

The protected content is decrypted by the CAS or DRM client in the CICAM, allowing the Host to present the content to the user.

4.3.3.3 Linear and VoD Streaming (push)

The content is streamed from the server using a push protocol such as RTSP/RTP. The Host is able to perform trick play, for example to skip forwards or backwards and fast forward and reverse modes within a content item, by using the RTSP protocol to send the appropriate commands.

The protected content is decrypted by the CAS or DRM client in the CICAM, allowing the Host to present the content to the user.

4.3.3.4 Downloaded content

This use case is possible only in IP delivery Host player mode. The content has previously been downloaded to local storage and is protected by DRM encryption. The content is now read by the Host. The Host might be able to perform trick play, for example to skip forwards or backwards and fast forward and reverse modes within a content item, by either reading the content faster than real time or by reading chunks of the content (e.g. I-frames).

The protected content is decrypted by the DRM client in the CICAM, allowing the Host to present the content to the user. Playback might be possible only when a connection can be made to the DRM server.

4.4 CI PlusTM browser extensions

The CI PlusTM browser is extended to include the functionality of static content retrieval over an IP connection by a CI PlusTM browser application, by adopting a profile of the MHEG interaction channel specified in the MHEG-5 Broadcast Profile specification [6]. This aspect is specified in clause 11.2 of the present document.

In CI PlusTM V1.3 [3] it is optional for the Host to support scaled video within a CI PlusTM browser application. The present document specifies that scaled video shall be supported in the CI PlusTM browser. The corresponding affected amendments for this aspect are specified in clause 11.1.5.

Host devices increasingly include hybrid broadcast broadband capability. The present document contains extensions to CI PlusTM that enable the CI PlusTM Browser to make use of this capability. Extensions are also defined to enable improvements in the user experience of operator applications using the CI PlusTM Browser, such as for VoD and EPG.

4.5 CICAM application launching

Host devices often include a middleware environment, for example HbbTV® or MHP®, to support broadcast and/or online interactive applications. The present document provides mechanisms for the CICAM to enquire about the support of middleware systems, also referred to as application environments, in the Host, and to launch applications to run on the Host middleware. The present document specifies only the framework within which applications for Host-supported environments other than the CI PlusTM browser can be launched from the CICAM. It is left to each application environment to define the identifier parameters necessary to be able to make use of this framework. These issues are contained in clause 12.

Clause 12 also defines an application coordination framework that addresses issues around application launch contention between CICAM applications and broadcast applications. The present document provides mechanism whereby the CICAM can offer a Virtual Channel that is added to the channel line-up in the Host, which launches a CICAM application when selected. The user can access this Virtual Channel using the normal methods available in the Host UI for channel selection in combination with a CICAM. The CICAM Virtual Channel mechanism is specified in clause 14.

4.6 CICAM file retrieval

The present document specifies a mechanism whereby a Host application, for example HbbTV® or MHP®, can retrieve files that are stored on the CICAM. The file system is elementary in nature - it is read-only and there is no notion of directory structure or discovery. The intended usage of this file retrieval facility is for applications running in the Host to read UI assets, for example icons or graphics, from the CICAM rather than from a server via the Host's IP connection. The application is assumed to know in advance which file assets are available on the CICAM. A request for a file will either succeed, or fail if the file is not available.

CICAM file retrieval is specified in clause 9.

4.7 Usage Rules Information extensions

The present document specifies an extension of the Content Control URI signalling to support trick mode inhibition, i.e. trick mode control operations to be enabled or disabled for the corresponding item of controlled content. This control applies to controlled content with copy control status of copy one generation only. The rules for interpretation of the trick mode inhibit URI in the Host are outside the scope of the present document.

The trick mode inhibit URI extension is introduced by defining a new version of Content Control URI, version 3, specified in clause 11.

4.8 Watermarking and transcoding

The primary application of the original Common Interface Specification EN 50221 [1] was for the descrambling of particular services or their component streams by a CA system embedded in the CICAM. Other applications, for example those which necessitate any manipulation of the contents of the service or one or more of its components being decrypted, were not foreseen. However, the present document specifically foresees CICAM applications that include the addition of a watermark on the service or one or more of its components being decrypted or otherwise processed, or the transcoding of the service or any of its components being received.

Digital watermarking provides a mechanism for deterring piracy by making it possible to identify the exact source of illegal copying, by embedding data into one or more component streams (compressed or uncompressed) in real time. The use case to be supported is watermarking on the compressed video on receiver side performed by the CICAM. On the server side, the content is prepared for watermark insertion. After reception and CA or DRM decryption, the CICAM adds some information specific to the individual CICAM (such as CAS serial number and timing information) in the place-holder prepared by the server.

Transcoding is relevant, for example in IP delivery CICAM player mode, where the Host does not support the codec used for one or more of the delivered service components. The CICAM transcodes each relevant component stream to a format that is supported by the Host.

In IP delivery Host player mode there is a severe constraint on the feasibility of watermarking and transcoding operations on content contained in the ISOBMFF file format. This is specified in clause 5.2.

The present document allows the CICAM to perform watermarking or transcoding operations on content or services that it is descrambling, without specifying the watermarking methods or transcoding formats. Normative statements about watermarking and transcoding operations performed in the CICAM are contained in clause 5.2.

5 General requirements

5.1 Backwards compatibility

Hosts that are compliant with the present document shall operate with CICAMs conforming to previous versions of CI PlusTM, i.e. V1.3 [3] and earlier, and with CICAMs conforming to DVB CI [1],[2].

Particular issues concerning multi-stream operation in the case when multiple CICAMs are installed are addressed in clause 6.2.7.

CICAMs that are compliant with the present document shall operate in Hosts conforming to previous versions of CI PlusTM, i.e. V1.3 [3] and earlier, and with Hosts conforming to DVB CI [1] and [2].

In each case the available functionality shall correspond to the mutual capabilities of the CICAM and Host.

5.2 Watermarking and transcoding

CI PlusTM CICAMs that are compliant with the present document may perform the following operations in addition to the descrambling of protected services or their component streams:

• the addition of a watermark;

• transcoding of one or more component streams.

Irrespective of the content processing function performed by the CICAM, a compliant TS shall be provided for the Host

Watermarks may be added only to content or services that are being descrambled by, or are associated with the CICAM. As stated in clauses 5.2 and 6.2 of [1], incoming packets to the CICAM that are not scrambled shall be returned to the Host unmodified. Hence watermarks shall not be applied to content or services that are not received in scrambled form.

Watermarking operations performed by the CICAM shall not result in any perceivable degradation in audio and video quality of the corresponding service after decoding by the Host.

Watermarking operations on ISOBMFF format IP-delivered content that imply a change in Sample size between input and output from the CICAM may be performed by the CICAM using IP delivery CICAM player mode only.

Transcoding operations may be performed by the CICAM using IP delivery CICAM player mode only. Such transcoding operations will likely result in a bitstream returned to the Host that has a significantly different bit rate compared to the input stream into the CICAM, hence the output clock may be at a significantly different frequency compared to the input clock. Transcoding operations may also result in a longer delay for streams being processed by the CICAM.

When performing watermarking or transcoding operations, the CICAM may insert new TS packets, delete some of the input packets and/or change the contents of the input TS packets.

5.3 PES level scrambling

The CI PlusTM V1.3 specification [3] makes provision for protected content delivered with scrambling applied at the PES level to be re-scrambled by the CICAM using the TS level scrambling method specified in that same specification.

Due to the security vulnerability introduced by such re-scrambling the present document specifically excludes that mode of operation.

6 Multi-stream reception

6.1 General

This clause specifies extensions to enable the carriage of multiple services from multiple tuners on the Host over the TS Interface with a single instance of the TS Interface and a single attached CICAM.

For multi-stream operation the TS Interface shall be put into a special mode of operation, called multi-stream mode, rather than the conventional TS Interface mode in accordance with DVB-CI [1], where services only from a single broadcast source may be decrypted.

In Multi-Stream Mode the Host sends one or more Local TSs to the CICAM, using the multi-stream facilities specified in the present clause.

Figure 3 depicts a schematic diagram of the extended CI PlusTM architecture to facilitate multi-stream support, as an informative example of a Host device with three broadcast tuners and IP delivery capability.

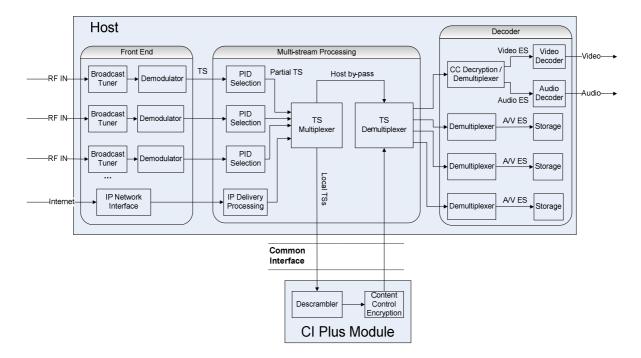


Figure 3: CI Plus™ extended architecture for multi-stream reception

In the context of the present document a tuner is defined to deliver an MPEG-2 TS over any of, but not limited to the DVB terrestrial, satellite, cable physical layers, or an IP network interface. In the case of the IP "tuner", as well as being able to receive a conventional CA-protected TS, content may be delivered also in ISOBMFF format or a derivative thereof, for example ISOBMFF contained in an MPEG DASH asset. A TS-based content item may also be contained in an MPEG DASH asset. The handling of IP-delivered content with CI PlusTM is described in clause 4.3, and specified in clause 7 (Host player mode) and clause 8 (CICAM player mode).

In this new architecture, each service to be decrypted by the CICAM may be first extracted from the incoming TS by selecting the PIDs associated with that service, to form a Local TS. Multiple Local TSs are multiplexed together before being sent to the CICAM, which decrypts each service independently. When the multiplexed Local TSs are returned decrypted to the Host, they are demultiplexed back to separate SPTSs, which can then be decoded by the Host. A multi-stream capable Host shall operate in accordance with DVB-CI [1] until the CICAM has completed the initialisation of the multi-stream resource, specified in clause 6.4.2 of the present document, by sending the CICAM_multistream_capability() APDU.

When a multi-stream resource session has been established, the CICAM shall send the CICAM_multistream_capability() APDU at the earliest time possible. Once the CICAM has sent the CICAM multistream capability() APDU, it shall be ready to receive one or more Local TSs in Multi-stream Mode.

Several CI [1] and CI PlusTM [3] resources have been upgraded to add the multi-stream context in the present document. A multi-stream Host shall offer both the multi-stream resource types and the conventional single-stream resource types to the CICAM. This enables a V1.3 CI PlusTM or earlier compliant CICAM to function normally in Single-stream Mode.

If the Host supports the multi-stream resource, the multi-stream CICAM may request a session to either the multi-stream types or the single-stream types of the relevant resources. If any of the multi-stream resource types are opened the CICAM shall continue to operate in Multi-stream Mode, meaning that the CICAM shall request sessions only to the multi-stream resource types, until all of the multi-stream resources types are closed.

If the single-stream types of the relevant resources are requested by the CICAM, then the Host shall operate in single-stream Mode according to DVB-CI [1].

If the multi-stream types of the relevant resources are requested by the CICAM, then the multi-stream capable Host shall operate in multi-stream mode as defined in the present document. While operating in multi-stream mode the Host may send a single Local TS to the CICAM.

In Multi-stream Mode only one service selected to be descrambled shall be carried in each Local TS. If two services originating from the same physical tuner are selected to be descrambled, then a separate Local TS shall be generated for each service.

The number of services that can be carried over the TS Interface in multi-stream mode is limited only by the overall bandwidth of the TS interface (96 Mbps) and the capabilities of both the Host and the CICAM. Because there is no standardised mechanism to determine the maximum bitrate of a broadcast service, it is left to the judgement of the Host implementation and knowledge of the available broadcast networks to decide how much bandwidth is allocated for each of the services carried in multi-stream mode, and at what point the Host needs to decide not to multiplex additional streams, due to the risk of exceeding the maximum total bitrate available on the TS Interface.

Clause 6.2.7 specifies additional rules for multi-stream operation when both a multi-stream CICAM and a non-multi-stream CICAM are installed in the multi-stream Host.

Clause 6.4 contains the specification of the new multi-stream resource.

Wherever necessary to enable multi-stream functionality, resource types specified in DVB-CI [1] and CI PlusTM V1.3 [3] have been updated with the addition of the LTS_id parameter, to identify the Local TS. These are specified in clauses 6.4 and onwards.

Annex A lists the resources that are required to support multi-stream functionality.

6.2 TS Interface and Local TS multiplexing

6.2.1 Local TS identifier

Each of the Local TSs is assigned a Local TS identifier (*LTS_id*) by the Host to identify it uniquely. The *LTS_id* replaces the fixed value of 0x47 in the sync byte field of the TS packet header of every TS packet passed from the Host to the CICAM. The Host shall set the sync byte field in the TS packet header to carry the *LTS_id* of each corresponding TS packet sent to the CICAM. The value of *LTS_id* shall be unique for each Local TS and shall not change while the Local TS is selected to be descrambled and while it is being sent over the TS interface. It is recommended that *LTS_id* values assigned by the Host start from 0x47, i.e. the *LTS_id* of the first Local TS is set to 0x47, the *LTS_id* of the second Local TS is set to 0x48, and so on.

Since the Local TS sync byte may deviate from the fixed standard value of 0x47, this recurring byte value shall not be relied upon for the detection of the first byte of each TS packet. The *MISTRT* and *MOSTRT* control signals of the TS Interface perform this function and shall be used to determine the start of a TS packet.

The CICAM shall not modify the LTS_id field of any incoming Local TS.

The *LTS_id* is used by the demultiplexer in the Host and CICAM to identify the TS packets associated with each Local TS. The Host may regenerate each Local TS into a fully compliant TS by replacing the *LTS_id* with the standard fixed sync byte value of 0x47 but the Host need not do so prior to subsequent decoding or storage, as required, if the resultant TS will not be used by devices or applications that require a fully compliant TS.

If the TS Interface is being operated in single-stream mode, then the *LTS_id* by implication shall have the value of 0x47.

6.2.2 Multiplexing broadcast and IP-delivered content

IP-delivered content shall be carried in TS format between the Host and CICAM and back to the Host. Clause 7.5.3 specifies the method to encapsulate non-TS format IP-delivered content into a TS for carriage over the TS Interface.

In Single-stream Mode the IP-delivered content is carried as the single TS over the TS Interface.

In multi-stream mode the IP-delivered content is carried as a Local TS, either alone or multiplexed with further broadcast or IP-delivered Local TSs as specified in the present clause.

6.2.3 Multiplexed TS packet order, delay and delay variation

The CI specification [1] sets out requirements of the CICAM concerning packet delay and delay variation for the single TS passing through the CICAM from and back to the Host. Table 1 specifies the requirements on the CICAM as regards the management of TS packet order, delay and delay variation for the various use cases applicable in the present document. The first use case, broadcast only, extends the requirements contained in the DVB-CI specification [1], for multi-stream mode as specified in the present document. The next two use cases apply to both IP delivery Host Player mode, specified in clause 7, and IP delivery CICAM Player mode, specified in clause 8, and the final use case applies to the combined carriage of broadcast and IP-delivered content over the TS Interface.

Delay Variation applied to Use Case Packet order **Packet Delay** input bytes CICAM shall conform to the CICAM shall conform to the CICAM shall maintain the constant delay requirement in maximum delay variation **Broadcast only** order of packets with same clause 5.4.2 of [1], when applied to any byte requirement LTS_id. processing an input transport in clause 5.4.2 of [1]. packet. CICAM need not maintain CICAM need not conform to the CICAM need not conform to the the order of packets with the **IP** delivery Host constant delay requirement in maximum delay variation same LTS_id. The order player mode only clause 5.4.2 of [1], when applied to any byte requirement shall be maintained for processing an input IP packet. in clause 5.4.2 of [1]. packets with the same PID. IP delivery Not applicable as CICAM Not applicable as CICAM Not applicable as CICAM **CICAM** player generates a new TS. generates a new TS. generates a new TS. mode only CICAM need not conform to the CICAM shall maintain the order of broadcast packets constant delay requirement in with same LTS id. clause 5.4.2 of [1], when CICAM need not conform to the **Broadcast and IP** CICAM need not maintain processing an input broadcast maximum delay variation delivery Host the order of IP delivery Host transport packet. applied to any byte requirement player mode player mode packets with the CICAM may not conform to the in clause 5.4.2 of [1]. same LTS_id. The order constant delay requirement in shall be maintained for clause 5.4.2 of [1], when packets with the same PID. processing an input IP packet. CICAM need not conform to the CICAM need not conform to the CICAM shall maintain the constant delay requirement in maximum delay variation order of broadcast packets clause 5.4.2 of [1], when **Broadcast and IP** applied to any byte requirement with same LTS_id. processing an input broadcast delivery CICAM in clause 5.4.2 of [1]. Not applicable to IP-delivered transport packet. player mode Not applicable to IP-delivered content as CICAM generates Not applicable to IP-delivered content as CICAM generates a a new TS. content as CICAM generates a new TS. new TS.

Table 1: TS packet order, delay and delay variation

6.2.4 Scrambling cipher and CCK usage

The Host and the CICAM shall follow the same scrambling cipher selection rules as in table 5.6 of the CI $Plus^{TM}$ V1.3 specification [3]. The same selected cipher shall be used on all of the LTSs.

Different key material Km, consisting of Content Control Key *CCK* and Content Initialisation Vector *CIV* (if required), shall be derived for each Local TS, based on a common Key Precursor Kp.

The function f-CC used to compute the Km per LTS shall be determined by the Trust Authority.

6.2.5 Host Service Shunning

When Host Service Shunning is active on a selected broadcast service (as defined in clause 10 of the CI PlusTM V1.3 specification [3]), the Host shall ensure that the shunned service is not provided to the CICAM for descrambling. In multi-stream mode the Host shall by-pass the multiplexing of the Local TS containing the shunned service into the multi-stream TS sent to the CICAM, i.e. Service Shunning shall be applied only to the service that the Service Operator has signalled to be shunned, and not to any other service, unless they also have signalling indicating that they are to be shunned.

Host Service Shunning is not applicable to either of the IP delivery modes described in clauses 7 and 8 of the present document.

6.2.6 TS clock

As a result of multiplexing variable bit rate Local TSs, the bit rate over the TS Interface may change continuously, hence the Host and CICAM shall support variable MPEG Input Clock and MPEG Output Clock frequencies, including times when the clock may stop completely.

6.2.7 Multi-stream operation with multiple CICAMs

As specified in clause 5.1, multi-stream Hosts shall be able to operate with CICAMs conforming to earlier versions of the CI PlusTM specification and with DVB-CI [1], but particular provisions are provided for the case when multiple CICAMs are installed in the multi-stream capable Host, as follows:

- Host with multi-stream CICAMs only: The Host may daisy-chain the multiplex of Local TSs through all of the CICAMs.
- Host with single-stream CICAMs only: The Host shall operate in Single-stream Mode, i.e. the Host may daisy-chain a single TS through all of the single-stream CICAMs.
- Host with both multi-stream and single-stream CICAMs installed: The Host may either:
 - continue to operate in Multi-stream Mode, and daisy-chain the multiplex of Local TSs only through all of the multi-stream capable CICAMs, and bypass all of the single-stream CICAMs; or
 - operate in Single-stream Mode, and daisy-chain a single TS through all of the CICAMs.

6.3 PID Selection

6.3.1 General

A broadcast TS generally contains more than one service. In order to reduce bandwidth over the TS Interface, for each service to be decrypted the Host may choose to select only the PIDs corresponding to the selected service from the tuner to be output to form a new partial TS, and may discard the remaining incoming PIDs.

For each selected service, if the Host applies PID selection as defined in the present clause, the Host shall maintain a list of selected PIDs. The initial contents of this list shall be determined by the Host and shall include the minimum default set of PIDs as specified in clause 6.3.2 or 6.3.3, as appropriate. The list may also include any PIDs that the Host may select for its own needs.

The selected PIDs list may then be updated further due to any of the following events:

- A user initiated (or other) change in the elementary stream selection, for example a change of active audio Track.
- A request from the CICAM to modify the list to add or remove PIDs. The request consists of a list of PIDs in decreasing order of priority.
- Any other event that requires the Host to modify the list of selected PIDs.

These scenarios are described in more detail in the following clauses.

6.3.2 Default PID selection

The following PIDs shall be selected by the Host by default upon a service selection and the corresponding TS packets shall be sent over the TS Interface:

- The ES PIDs (Elementary_PIDs) that are declared in the corresponding *ca_pmt()* APDU.
- The CA_PIDs declared in a CA_descriptor() present in the corresponding ca_pmt() APDU.

- The PID containing the PMT of the selected service.
- The EIT PID, as defined in EN 300 468 [10].
- The SDT PID, as defined in EN 300 468 [10].

If two or more services are selected for decryption from the same tuner then the SI PIDs shall be duplicated in each of the partial TSs generated for each service.

The Host may include additional PIDs in the partial TS.

6.3.3 Default PID selection for frequency tune

In the particular case where the CICAM issues a frequency tune by use of the *tune_broadcast_req()* APDU without referring to a particular service, by setting the *service_id* field in the APDU to the value 0x0000, then the default selected PID list shall contain:

- The PAT PID, as defined in MPEG-2 Systems [4].
- The CAT PID, as defined in MPEG-2 Systems [4].
- The EIT PID, as defined in the DVB SI specification [10].
- The SDT PID, as defined in the DVB SI specification [10].
- The NIT PID, as defined in the DVB SI specification [10].

The Host may include additional PIDs in the Local TS.

6.3.4 PID selection priority

The CICAM shall provide the list of PIDs that it wants to receive for the selected service in decreasing order of priority. The highest priority PIDs shall be those that are critical for descrambling. This is an attribute that may be set for each requested PID in the $PID_select_req()$ APDU. The Host shall select PIDs in order of priority following the order of PIDs in the list. If the Host is not able to include one or more of the PIDs that are indicated by the CICAM to be critical for descrambling (in the $PID_select_req()$ APDU) then the CICAM might not be able to descramble the service correctly.

6.3.5 CICAM initiated update

The CICAM may request an update of the Host selected PIDs list. The *PID_select_req()* APDU shall be used to modify the list of PIDs requested by the CICAM. This list may include PIDs that are not referenced in the PMT of the selected service. The Host shall respond to such a request using the *PID_select_reply()* APDU. The set of ES PIDs for the service, i.e. those listed in the *ca_pmt()* cannot be altered by the CICAM.

The resulting PID list then contains the union of:

- ES PIDs (Elementary_PIDs) declared in the most recent occurrence of the *ca_pmt()* for the service.
- The PIDs declared by the CICAM in the *PID_select_reg()* APDU.
- Any additional PIDs selected by the Host for its own needs.

If the Host is not able to select all of the above PIDs, then the Host shall select PIDs in the following priority order. The first set has the highest priority:

- a) The ES PIDs (*Elementary_PIDs*) declared in the latest occurrence of the *ca_pmt()* related to the service.
- b) The PIDs requested by the CICAM, in order of priority, given by their order in the list, and the additional PIDs selected by the Host for its own needs. It is up to the Host implementation how to prioritise its own set of needed PIDs among the list of additional PIDs selected by the CICAM.

6.3.6 Change in ES selection

Whenever the Host sends an update of the $ca_pmt()$ for a selected service ($ca_pmt_list_management = 0x05$) the Host shall revert to the default set of PIDs as defined in clause 6.3.2.

6.3.7 Host initiated PID addition or removal

The Host may add or remove PIDs to and from the previously selected PID list for its own needs. The adding of a PID by the Host may result in the de-selection of a PID previously requested by the CICAM. In this case the Host shall inform the CICAM of this by sending the *PID_select_reply()* APDU accordingly.

Although some of the PIDs requested by the CICAM might not be critical for descrambling, they should be granted and kept in the Local TS as far as this is within the capabilities of the Host.

Conversely, as a consequence of the Host de-selecting one of its own selected PIDs, the Host may re-select one of the CICAM requested PIDs and inform the CICAM of this by sending the *PID_select_reply()* APDU accordingly. The Host need not inform the CICAM if it adds or removes PIDs that were not on the list of PIDs requested by the CICAM.

The Host shall consider the priority order established by the CICAM when de-selecting or re-selecting a PID.

6.3.8 Selected services from the same broadcast TS

If two or more services are selected from the same broadcast TS and there are PIDs that are common between multiple services, then the TS packets matching the common PIDs shall be duplicated in the Local TS generated for those services.

6.3.9 Example PID selection sequence

Figure 4 shows an example sequence diagram of how the CICAM may use the *PID_select_req()* APDU for updating the selected PID list.

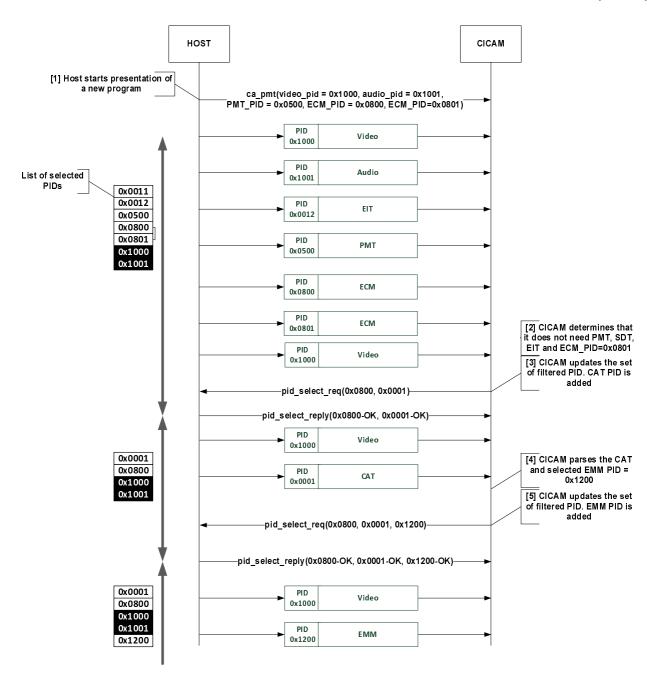


Figure 4: PID selection example sequence diagram

- 1) The Host selects a new service. The *video_pid* 0x1000 and *audio_pid* 0x1001 are scrambled and hence included in the *ca_pmt* as *elementary_PID* and selected by the Host. The PMT PID of the service is 0x0500 and is selected by the Host. The PMT contains two *ca_descriptors* matching the *ca_system_id* of the CICAM. The corresponding PIDs (0x0800, 0x0801) are selected by the Host. The default list of selected PIDs is then {0x0011, 0x0012, 0x0500, 0x0800, 0x0801, 0x1000, 0x1001}.
- 2) The CICAM determines that it does not need the Host to filter SDT PID, EIT PID, PMT PID and the ECM PID 0x0801.
- 3) The CICAM also needs the CAT PID to be included in the selected PID list. The CICAM then sends a *PID_select_req()* APDU including the remaining ECM PID (0x0800) and the CAT PID (0x0001). The Host replies positively and the new list of selected PIDs is then {0x0001, 0x0800, 0x1000, 0x1001}.
- 4) The CICAM receives the first CAT, parses it and determines the required EMM PID (0x1200).

5) The CICAM then sends a *PID_select_req()* APDU including the remaining ECM PID (0x0800), the CAT PID (0x0001) and the EMM PID (0x1200). The Host replies positively and the new list of selected PIDs is then {0x0001, 0x0800, 0x1000, 0x1001, 0x1200}.

6.4 Resources for multi-stream operation

6.4.1 General

One new resource is defined for multi-stream operation, namely the *multistream* resource. This is specified in clause 6.4.2.

Further resources needed for multi-stream operation are defined as new types of the following existing resources, and are specified in clauses 6.4.3 and onwards:

- Content Control;
- Host Control;
- Conditional Access Support;
- MMI; and
- Application MMI.

Annex B provides an overview of the new datatype identifiers for multi-stream functionality.

6.4.2 Multi-stream resource

6.4.2.1 General

The *multistream* resource contains an APDU for the CICAM to indicate its multi-stream related capabilities, and APDUs to manage PID selection in the Local TSs. Hosts and CICAMs supporting the multi-stream feature shall implement the *multistream* resource.

Table 2 contains a summary of the *multistream* resource.

Table 2: Multi-stream resource summary

Resource				Application Object		Direction		
Name	Resource Identifier	Class	Туре	Ver.	APDU Tag	Tag value	Host	CICA M
Multistream	00 90 00 41	144	1	1	CICAM_multistream_capability	9F 92 00		-
					PID_select_req	9F 92 01		-
					PID_select_reply	9F 92 02		\rightarrow

The remainder of clause 6.4.2 specifies the APDUs defined for the multistream resource.

6.4.2.2 CICAM multi-stream capability APDU

The CICAM shall use the CICAM_multistream_capability() APDU to indicate its multi-stream capabilities to the Host. The multi-stream capabilities include the maximum numbers of TS and ES that the CICAM is able to descramble concurrently.

Based on the CICAM capabilities, the Host shall limit both the number of Local TSs that it concurrently multiplexes and the number of ESs that it requests the CICAM to descramble, so as not to exceed the indicated maximum values supported by the CICAM.

The CICAM shall notify the Host of its multi-stream capabilities when a multistream resource session is opened.

The CICAM_multistream_capability() APDU is sent from the CICAM to the Host. Table 3 shows the syntax of the CICAM_multistream_capability() APDU.

Table 3: CICAM_multi-stream_capability APDU

Syntax	Number of bits	Mnemonic
CICAM_multistream_capability () {		
CICAM_multistream_capability_tag	24	uimsbf
length_field ()		
max_local_TS	8	uimsbf
max_descramblers	16	uimsbf
}		

The fields are defined as follows:

CICAM_multistream_capability_tag: This 24 bit field with value 0x9F9200 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

max_local_TS: The maximum number of Local TSs that the CICAM is able to receive concurrently.

max_descramblers: The total number of descramblers that the CICAM is able to provide concurrently for all Local TSs.

6.4.2.3 PID select request APDU

A request for PIDs is sent from the CICAM to the Host. It is used by the CICAM to request a specific set of PIDs to be included in the received Local TS.

The CICAM shall provide a list of PIDs in order of priority following the order of PIDs in the list. Each PID in the list is accompanied by the *critical_for_descrambling_flag*, which the CICAM shall set for PIDs that are necessary to be able to descramble the content. The list of PIDs shall not contain any PIDs with the *critical_for_descrambling_flag* set to '1' after the first PID that has the *critical_for_descrambling_flag* set to '0'.

If the Host is not able to include one or more of the PIDs that have the *critical_for_descrambling_flag* set then the CICAM might not be able to descramble the service correctly.

At any point in time, the Host may deselect or reselect one of the CICAM selected PIDs. When this occurs, the Host shall inform the CICAM by sending a *PID_select_reply()* APDU with updated status for those PIDs that are no longer selected, or selected again.

If it is necessary for the Host to deselect one or more PIDs, it shall choose the lowest priority PIDs from the list provided by the CICAM when the PIDs were selected.

Conversely, if the Host has the possibility to reselect one or more PIDs that had been deselected, it shall choose the highest priority PIDs from the list provided by the CICAM when the PIDs were requested initially.

If two or more services are selected for decryption from the same tuner and the CICAM selected PID list have common PIDs, then the transport packets matching the common PIDs shall be duplicated in each of the Local TSs generated for each service.

Table 4: PID_select_req APDU

Syntax	Number of bits	Mnemonic
PID_select_req () {		
PID_select_req_tag	24	uimsbf
length_field ()		
LTS_id	8	uimsbf
num_PID	8	uimsbf
for $(i = 0; i < num_PID; i++)$ {		
reserved	2	bslbf
critical_for_descrambling_flag	1	bslbf
PID	13	uimsbf
}		
}		

The fields are defined as follows:

PID select req tag: This 24 bit field with value 0x9F9201 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS id: Local TS identifier.

num_PID: Number of PIDs contained in the following loop.

critical_for_descrambling_flag: When set to '1', the associated PID is critical for descrambling. When set to '0', the associated PID is not critical for descrambling.

PID: Requested PID value. The Host shall ignore any request for a PID value of 0x1FFF.

6.4.2.4 PID select reply APDU

When the Host receives a PID request from the CICAM, the Host shall acknowledge the request by confirming whether it was able to enable the selection of the requested PIDs, using the *PID_select_reply()* APDU. The Host shall indicate the status for all of the PIDs that were requested for selection in the previous *PID_select_req()* APDU by setting the *PID_select_flag* field accordingly.

The CICAM shall wait for the reception of the *PID_select_reply()* APDU before issuing the next *PID_select_req()* APDU for the same *LTS_id*.

Whenever the Host de-selects or re-selects one or more PIDs requested by the CICAM, the Host shall send the *PID_select_reply()* APDU to inform the CICAM of the change. The *PID_select_reply()* APDU shall indicate the status for all of the PIDs that were present in the previous *PID_select_req()* APDU.

Number of bits Mnemonic **Syntax** PID_select_reply () { PID_select_reply_tag 24 uimsbf length_field () 8 LTS id uimsbf 7 reserved uimsbf PID_selection_flag 1 uimsbf num_PID 8 uimsbf for $(i=0; i < num_PID; i++)$ reserved 2 bslbf PID_selected_flag uimsbf 1 PID 13 uimsbf

Table 5: PID select reply APDU

The fields are defined as follows:

PID_select_reply_tag: This 24 bit field with value 0x9F9202 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_id: Local TS identifier.

PID_selection_flag: The status, if PID selection has been applied for the Local TS. If the whole TS is sent as the Local TS then this value shall be set to 0b0 and the *num_PID* field shall be set to 0x0. If PID selection is applied this value shall be set to 0b1 and the Host shall inform the CICAM about the selected PIDs using the *num_PID* field and the list of selected PIDs.

num_PID: Number of PIDs contained in the following loop.

PID_selected_flag: The status of the corresponding requested PID. A PID that could be selected successfully shall have this bit set to 0b1. If the PID could not be selected by the Host then its value shall be 0b0.

PID: PID value for which the *PID_selected_flag* field applies.

6.4.3 Content Control resource

6.4.3.1 General

In order to support multi-stream reception, the Content Control resource is extended with a new resource type (0x008C1041). The constituent APDUs of the Content Control resource are specified in the following clause.

Table 6 contains a summary of the Content Control resource.

Table 6: Content Control resource summary

	Resc	ource			Application Obj	ect	Direction
Name	Resource Identifier	Class	Type	Ver.	APDU Tag	Tag value	Host CICA
Content	00 8C 10	140	65	1	cc_open_req	9F 90 01	-
Control	41				cc_open_cnf	9F 90 02	\rightarrow
					cc_data_req	9F 90 03	+
					cc_data_cnf	9F 90 04	\rightarrow
					cc_sync_req	9F 90 05	+
					cc_sync_cnf	9F 90 06	\rightarrow
					cc_sac_data_req (note 1)	9F 90 07	$\leftarrow \rightarrow$
					cc_sac_data_cnf (note 1)	9F 90 08	$\leftarrow \rightarrow$
					cc_sac_sync_req	9F 90 09	←
					cc_sac_sync_cnf	9F 90 10	\rightarrow
					cc_PIN_capabilities_req	9F 90 11	\rightarrow
					cc_PIN_capabilities_reply	9F 90 12	←
					cc_PIN_cmd	9F 90 13	\rightarrow
					cc_PIN_reply (note 2)	9F 90 14	←
					cc_PIN_event (note 2)	9F 90 15	←
					cc_PIN_playback	9F 90 16	\rightarrow
					cc_PIN_MMI_req	9F 90 17	\rightarrow

NOTE 1: The APDU syntax is not extended but the *LTS_id* field is included in certain SAC protocols, as specified in clause 6.4.3.3.

NOTE 2: This APDU is extended to include the LTS_id field.

The remainder of clause 6.4.3 specifies the APDUs defined for the multi-stream extension of the Content Control resource that are extended to include the Local TS identifier (*LTS_id*) field. The remaining APDUs in this resource type are unchanged in their syntax from that defined in CI PlusTM V1.3 [3], clauses 11.3.1 and 11.3.2.

Clause 6.4.3.3 specifies extensions to the existing Content Control protocols specified in CI Plus™ V1.3 [3].

6.4.3.2 Content Control APDU extensions

6.4.3.2.1 cc_PIN_reply APDU

The $cc_PIN_reply()$ APDU, used for the record start protocol, is extended to include LTS_id , as shown in table 7. The Host shall use this extended APDU while multi-stream mode is active.

Table 7: cc_PIN_reply APDU syntax

Syntax	Number of bits	Mnemonic
cc_PIN_reply() {		
cc_PIN_rely_tag	24	uimsbf
length_field()		
reserved	7	uimsbf
LTS_bound_flag	1	uimsbf
<pre>if (LTS_bound_flag == 1) {</pre>		
LTS_id	8	uimsbf
} else {		
reserved	8	uimsbf
}		
PINcode_status_field	8	uimsbf
]}		

The fields are defined as follows:

cc_PIN_reply_tag: This 24 bit field with value 0x9F9014 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_bound_flag: This 1-bit flag indicates that the $cc_PIN_reply()$ APDU is associated with a particular Local TS when set to 0b1. When set to 0b0 the $cc_PIN_reply()$ APDU is not associated with an LTS_id , for example when sent in response to the $cc_PIN_cmd()$, $cc_PIN_playback()$ or $cc_PIN_MMI_reg()$ APDUs.

LTS_id: Local TS identifier.

PINcode_status_field: Refer to clause 11.3.2.3 of the CI PlusTM V1.3 specification [3].

6.4.3.2.2 cc PIN event APDU

The *cc_PIN_event()* APDU, used for the record start protocol, is extended to include *LTS_id*, as shown in table 8. The Host shall use this extended APDU while multi-stream mode is active.

Number of bits Mnemonic cc PIN event() cc_PIN_rely_tag 24 uimsbf length field() LTS id 8 uimsbf program_number 16 uimsbf PINcode_status_field uimsbf 8 uimsbf pin_event_time_utc 40 uimsbf pin event time centiseconds 8 uimsbf private data 8x15 uimsbf

Table 8: cc PIN event APDU syntax

The fields are defined as follows:

- cc_PIN_event_tag: This 24 bit field with value 0x9F9015 identifies this APDU.
- **length_field:** Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.
- LTS id: Local TS identifier.
- Other fields: Refer to clause 11.3.2.4 of the CI PlusTM V1.3 specification [3].

6.4.3.3 Content Control protocol extensions

6.4.3.3.1 URI transmission and acknowledgement protocol

Table 9 shows the modified URI Transmission and Acknowledgement protocol as extended for multi-stream usage.

The CICAM shall wait for the acknowledgement from the Host before sending any further URI Transmission and Acknowledgement protocol message.

Table 9: URI transmission and acknowledgement protocol

Step	Action	APDU	Content		
1	CICAM sends the URI	cc_sac_data_req	send_datatype_nbr	r = 3	
	to the Host		1	datatype_id	datatype_len
			0	25 (uri_message)	64 bits
			1	26 (program_number)	16 bits
			2	50 (LTS_id)	8 bits
			request_datatype_nbr = 1		
			I	datatype_id	
			0	27 (uri_confirm)	
2	Host sends	cc_sac_data_cnf	send_datatype_nbr = 1		
	acknowledgement to		I	datatype_id	datatype_len
	the CICAM		0	27 (uri_confirm)	256 bits

6.4.3.3.2 Record Start protocol

Table 10 shows the modified Record Start protocol as extended for multi-stream usage.

The Host shall wait for the acknowledgement from the CICAM before sending any further Record Start protocol message.

Table 10: Record Start protocol

Step	Action	APDU	Content		
1	Host informs CICAM	cc_sac_data_req	send_datatype_nbr = 3 or 4		
	of start of recording		i	datatype_id	datatype_len
			0	38 (operating_mode)	8 bits
			1	26 (program_number)	16 bits
			2	39 (PINcode data)	variable (optional)
			3	50 (LTS_id)	8 bits
			request_datatype_n	br = 1	
			i	datatype_id	
			0	40 (record_start_status)	
2	CICAM sends	cc_sac_data_cnf	send_datatype_nbr	= 1	
	acknowledgement to		i	datatype_id	datatype_len
	the Host		0	40 (record_start_status)	8 bits

As a consequence of the execution of the Record Start protocol, the Host indicates whether the selected programme is user attended or not. If the selected programme is marked as unattended then the CICAM shall refrain from using the High Level MMI or Application MMI for user dialogue related to the selected program. The CICAM shall consider the selected programme as unattended when the operating_mode is set equal to either 0x01 (Timeshift) or 0x02 (Unattended_Recording).

6.4.3.3.3 Record Stop protocol extension

Table 11 shows the modified Record Stop protocol as extended for multi-stream usage.

The Host shall wait for the acknowledgement from the CICAM before sending any further Record Stop protocol message.

Table 11: Record Stop protocol

Step	Action	APDU	Content			
1	Host informs CICAM	cc_sac_data_req	send_datatype_nbr	send_datatype_nbr = 2		
	recording has		i	datatype_id	datatype_len	
	stopped		0	26 (program_number)	16 bits	
			1	50 (LTS_id)	8 bits	
			request_datatype_nbr = 1			
			i	datatype_id		
			0	42 (record_stop_status)		
2	CICAM sends a	cc_sac_data_cnf	send_datatype_nbr = 1			
	acknowledgement to		i	datatype_id	datatype_len	
	the Host		0	42 (record_stop_status)	8 bits	

6.4.3.3.4 Change Operating Mode protocol extensions

Table 12 shows the modified Change Operating Mode protocol as extended for multi-stream usage.

The Host shall wait for the acknowledgement from the CICAM before sending any further Change Operating Mode protocol message.

Table 12: Change Operating Mode protocol

Step	Action	APDU	Content		
1	Host informs CICAM	cc_sac_data_req	send_datatype_nbr	= 3	
	of change of		i	datatype_id	datatype_len
	operating mode		0	38 (operating_mode)	8 bits
			1	26 (program_number)	16 bits
			2	50 (LTS_id)	8 bits
			request_datatype_n	br = 1	
			i	datatype_id	
			0	41 (mode_change_status)	
2	CICAM sends a	cc_sac_data_cnf	send_datatype_nbr = 1		
	acknowledgement to		i	datatype_id	datatype_len
	the Host		0	41 (mode_change_status)	8 bits

6.4.3.3.5 CICAM to Host License Exchange protocol extensions

Table 13 shows the modified CICAM to Host License Exchange protocol as extended for multi-stream usage.

The CICAM shall wait for the acknowledgement from the Host before sending any further CICAM to Host License Exchange protocol message.

Table 13: CICAM to Host License Exchange protocol

Action	APDU	Content			
CICAM supplies the	cc_sac_data_req	send_datatype_nbr = 4 or 5			
Host with content		i	datatype_id	datatype_len	
license		0	26 (program_number) (see note 1)	16 bits	
		1	34 (license_status) (see note 2)	8 bits	
		2	25 (uri_message)	64 bits	
		3	33 (cicam_license)	variable	
		4	50 (LTS_id)	8 bits	
		request_	datatype_nbr = 1		
		I	datatype_id		
		0	35 (license_rcvd_status) (see note 3)		
Host confirms receipt	cc_sac_data_cnf	send_datatype_nbr = 1			
		I	datatype_id	datatype_len	
		0	35 (license_rcvd_status)(see note 3)	8 bits	
				ng of this field.	
	CICAM supplies the Host with content license Host confirms receipt 1: The program_numb 2: Table 11.45 in the 0	CICAM supplies the Host with content license	CICAM supplies the Host with content license CC_sac_data_req send_da	CICAM supplies the Host with content license CC_sac_data_req Send_datatype_nbr = 4 or 5 i datatype_id 0 26 (program_number) (see note 1) 1 34 (license_status) (see note 2) 2 25 (uri_message) 3 33 (cicam_license) 4 50 (LTS_id) request_datatype_nbr = 1 I datatype_id 0 35 (license_rcvd_status) (see note 3) Host confirms receipt CC_sac_data_cnf Send_datatype_nbr = 1 I datatype_id 0 35 (license_rcvd_status) (see note 3) 1: The program_number matches the Record Start message's program_number. 2: Table 11.45 in the CI Plus™ V1.3 specification [3] contains the allowed values and meaning the start message in the content of the conten	

NOTE 3. Table 11.42 in the CI Plus™ V1.3 specification [3] contains the allowed values and meaning of this field.

6.4.4 Conditional Access Support resource

6.4.4.1 General

In order to support multi-stream functionality, the Conditional Access Support resource has a new resource_type defined (resource_type = 2, version = 1), in which the ca_pmt() and ca_pmt_reply() APDU objects are extended by adding the Local TS identifier (LTS_id) to the APDU syntaxes. The ca_pmt() APDU is extended also with the PMT_PID field, in order to save the CICAM the task of parsing the Local TS to obtain it.

The Host and CICAM shall use these extended APDUs while multi-stream mode is active.

6.4.4.2 ca pmt APDU

The ca_pmt() APDU is extended to include the Local TS identifier and the PMT_PID fields as shown in table 14.

Table 14: ca_pmt APDU syntax

Syntax	No. of bits	Mnemonic
ca_pmt () {		
ca_pmt_tag	24	uimsbf
length_field ()		
LTS_id	8	uimsbf
ca_pmt_list_management	8	uimsbf
program_number	16	uimsbf
reserved	3	bslbf
PMT_PID	13	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
reserved	4	bslbf
program_info_length	12	uimsbf
if (program_info_length != 0) {		
ca_pmt_cmd_id /* at program level */	8	uimsbf
for $(i = 0; i < n; i++)$ {		
ca_descriptor() /* CA descriptor at programme level */		
}		
}		
for $(i = 0; i < n; i++)$ {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID /* elementary stream PID */	13	uimsbf
reserved	4	bslbf
es_info_length	12	uimsbf
if (es_info_length != 0) {		
ca_pmt_cmd_id /* at ES level */	8	uimsbf
for $(i = 0; i < n; i++)$ {		
ca_descriptor () /* CA descriptor at elementary stream level */		
}		
}		
}		

LTS_id: Local TS identifier.

ca_pmt_list_management: This parameter is used in single-stream mode to indicate whether the user has selected a single programme (comprised of one or several ES) or several programmes. In multi-stream mode each programme shall appear on a separate Local TS, thus in this case the values listed in table 15 may be used. These are a subset of those defined in clause 8.4.3.4 of the DVB-CI specification [1].

Table 15: ca pmt list management

ca_pmt_list_management	Value	
Only	0x03	
Update	0x05	
Reserved	Other values	

In multi-stream mode "Only" is used to start a new programme in the associated Local TS. This operation does not affect other Local TSs that may be running.

PMT_PID: The PID of the PMT of the selected service. Whenever the PMT PID of the selected service changes, a *ca_pmt()* APDU with the *ca_pmt_list_management* field set to 0x05 (update) shall be sent by the Host.

Other Fields: Refer to Table 25 of the DVB-CI specification [1].

6.4.4.3 ca_pmt_reply APDU

Table 16 shows the syntax of the *ca_pmt_reply* object for multi-stream operation, whereby the Local TS identifier (*LTS_id*) field is added.

Table 16: ca_pmt_reply APDU syntax

Syntax	No. of bits	Mnemonic
ca_pmt_reply () {		
ca_pmt_reply_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
ca_enable_flag	1	bslbf
if (ca_enable_flag == 1) {		
ca_enable /* at programme level */	7	uimsbf
} else {		
reserved	7	bslbf
}		
for $(i = 0; i < n; i++)$ {		
reserved	3	bslbf
elementary_PID	13	uimsbf
ca_enable_flag	1	bslbf
if (ca_enable_flag == 1) {		
ca_enable /* at elementary stream level */	7	uimsbf
} else {		
reserved	7	bslbf
}		
}		
}		

The fields are defined as follows:

LTS_id: Local TS identifier.

Other fields: Refer to Table 26 of the DVB-CI specification [1].

6.4.5 Multi-stream Host Control resource

6.4.5.1 General

The multi-stream Host Control resource type (with resource ID 0x00200081) is based on DVB Host Control version 3 as defined in clause 12. This multi-stream version of the resource allows the CICAM to request either the usual foreground tuning, i.e. for presentation to the user, or background tuning, meaning that the stream is not for presentation. The Host responds to a tune request, if successful, with the information of with which *LTS_id* the requested stream will be sent.

If the Host grants a background tune to the CICAM then the CICAM shall reply to an $ask_release()$ APDU within one second with the response "Release_OK" in the $ask_release_reply()$ APDU and shall release this resource without using Application MMI or High Level MMI, and it shall not affect any foreground stream.

The multi-stream Host Control resource supports the APDUs listed in table 17. All other APDUs from DVB Host Control resource V3 are not supported.

Table 17: Multi-stream Host Control APDUs

APDU	Direc	Direction	
	Host	CICAM	
tune_broadcast_req	-	-	
tune_triplet_req	-	-	
tune_lcn_req	-	-	
tune_ip_req	-	←	
tune_reply	7	\rightarrow	
ask_release	7	\rightarrow	
ask_release_reply	+		
tuner_status_req	←	←	
tuner_status_reply	\rightarrow		

The CICAM shall not request a tune that is to be presented and a background tune in the same session. The CICAM shall request another session to the Host Control resource if it requires both kinds of tuning operations, i.e. to be presented, and a background tune. A multi-stream Host shall support at least two sessions of the multi-stream DVB Host Control resource.

Within a given multi-stream Host Control session, any tune request overrides and replaces the previous request in the same session.

For multi-stream operation PID filtering may be required in order to keep the bandwidth requirement over the TS Interface below the maximum capacity. After a successful tune request has been acknowledged with the *tune_reply()* APDU, the Host shall provide the minimum PID subset depending on the tune request type. Where the tune request from the CICAM is to a service, the Host shall provide the minimum PID subset as defined in clause 6.3.2. The CICAM may then request additional PIDs using the *pid_select_req()* APDU.

Where the tune request from the CICAM is not to a service and just to a frequency, the Host shall provide the minimum PID subset as defined in clause 6.3.3. The CICAM may then request additional PIDs using the $pid_select_req()$ APDU. The syntax of the $tune_broadcast_req()$, $tune_triplet_req()$, $tune_lcn_req()$, $tune_ip_req()$ and $tune_reply()$ APDUs are modified in their multi-stream versions to allow the CICAM to request either the usual foreground tuning, i.e. for presentation to the user, or background tuning, meaning that the stream is not for presentation. All other APDUs retain the same syntax as in the DVB Host Control version 3 resource, specified in clause 12. The following clauses specify the modified syntaxes of the relevant APDUs for multi-stream operation.

6.4.5.2 tune broadcast reg APDU

The *tune_broadcast_request()* APDU version for multi-stream adds the facility of allowing the CICAM to indicate if the tune request is to be presented to the user or not. The Host shall respond with the *tune_reply()* APDU, informing the CICAM on which *LTS* id the requested tuned stream will be sent.

Table 18: tune_broadcast_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>tune_broadcast_req() { tune_broadcast_req_tag length_field() reserved background_tune_flag tune_quietly_flag keep_app_running_flag pmt_flag service_id reserved descriptor_loop_length for (i=0; i<n; descriptor()="" i++)="" pre="" {="" }="" }<=""></n;></pre>	24 4 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1	wimsbf uimsbf
<pre>if (pmt_flag == 1) {</pre>		

background_tune_flag: This field specifies if the tune request is to be performed in the background or if it is to be presented to the user. A value of 0b0 indicates that the tune is to be presented to the user; a value of 0b1 indicates that the tune is to be performed in the background, i.e. not to be presented to the user.

If the tune request is a background request, then the tune_quietly_flag and keep_app_running_flag shall be ignored.

Other fields: Refer to clause 12.2.1.

6.4.5.3 tune_triplet_req APDU

The *tune_triplet_req()* APDU version for multi-stream adds the facility of allowing the CICAM to indicate if the tune request is to be presented to the user or not. The Host shall respond with the *tune_reply()* APDU, informing the CICAM on which *LTS_id* the requested tuned stream will be sent.

Table 19 shows the syntax of the *tune_triplet_req()* APDU.

Table 19: tune_triplet_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>tune_triplet_req () {</pre>		
tune_triplet_req_tag	24	uimsbf
length_field()		
reserved	5	uimsbf
background_tune_flag	1	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
original_network_id	16	uimsbf
transport_stream_id	16	uimsbf
service_id	16	uimsbf
delivery_system_descriptor_tag	8	uimsbf
<pre>if (delivery_system_descriptor_tag == 0x7f) {</pre>		
descriptor_tag_extension	8	uimsbf
}		
else {		
reserved	8	uimsbf
}		
]}		

The fields are defined as follows:

background_tune_flag: See clause 6.4.5.2.

Other fields: Refer to clause 12.2.2.

6.4.5.4 tune_lcn_req APDU

The *tune_lcn_request()* APDU version for multi-stream adds the facility of allowing the CICAM to indicate if the tune request is to be presented to the user or not. The Host shall respond with the *tune_reply()* APDU, informing the CICAM on which *LTS_id* the requested tuned stream will be sent.

Table 20: tune_lcn_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>tune_lcn_req () {</pre>		
tune_lcn_req_tag	24	uimsbf
<pre>length_field()</pre>		
reserved	7	uimsbf
background_tune_flag	1	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
logical_channel_number	14	uimsbf

background_tune_flag: Refer to clause 6.4.5.2.

Other Fields: Refer to clause 12.2.3.

6.4.5.5 tune_ip_req APDU

The *tune_ip_request()* APDU version for multi-stream adds the facility of allowing the CICAM to indicate if the tune request is to be presented to the user or not. The Host shall respond with the *tune_reply()* APDU, informing the CICAM on which *LTS_id* the requested tuned stream will be sent.

Table 21: tune_ip_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>tune_ip_req () {</pre>		
tune_ip_req_tag	24	uimsbf
length_field()		
reserved	1	uimsbf
background_tune_flag	1	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
service_location_length	12	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
service_location_data	8	uimsbf
}		
]}		

background_tune_flag: Refer to clause 6.4.5.2.

Other Fields: Refer to clause 12.2.4.

6.4.5.6 tune_reply APDU

The *tune_reply()* APDU is sent by the Host in response to each of the three preceding tune request types specified for the DVB multi-stream Host Control resource. It is used to inform the CICAM if the tune was successful and with which *LTS_id* the requested stream will be sent.

Table 22: tune_reply APDU syntax

Syntax	Number of bits	Mnemonic
tune_reply () {		
tune_reply_tag	24	uimsbf
length_field ()		
LTS_id	8	uimsbf
status_field	8	uimsbf
}		

LTS_id: this field indicates the Local TS identifier where the Local TS resulting from the tune request can be found. The *LTS_id* shall be ignored if the tune request is unsuccessful.

Other Fields: Refer to table 14.30 of CI PlusTM V1.3 [3]].

6.4.6 Application MMI resource

6.4.6.1 General

In order to support multi-stream functionality, the Application MMI resource has a new resource type defined (resource_type = 2, version = 1), in which the RequestStart() APDU is extended by adding the Local TS identifier (LTS_id) to the APDU syntaxes. The Host may use the additional LTS_id in order to associate the request with a display in the situation where the Host is displaying more than one program at a time (e.g. picture-in-picture, mosaic or dual-display function).

The multi-stream capable type of the Application MMI resource specified in the present clause is based on the version 3 of type 1 Application MMI resource specified in clause 12.3, i.e. it also includes the ADQ option that is defined in clause 12.3.

The CICAM shall use this extended APDU while multi-stream mode is active.

6.4.6.2 RequestStart APDU

The RequestStart() APDU is extended to include the Local TS identifier as shown in table 23.

Syntax Number of bits Mnemonic RequestStart(){ RequestStartTag 24 uimsbf length field() reserved bslbf LTS bound flag bslbf 1 if (LTS bound flag == 1) { LTS_id 8 uimsbf else { reserved bslbf 8 AppDomainIdentifierLength 8 uimsbf InitialObjectLength uimsbf R for (i=0; i<AppDomainIdentifierLength; i++) {</pre> AppDomainIdentifier 8 uimsbf for (i=0; i < InitialObjectLength; i++) {</pre> InitialObject 8 uimsbf

Table 23: RequestStart APDU syntax

The fields are defined as follows:

LTS_bound_flag: This 1-bit flag indicates that the request is associated to a particular Local TS when set to "1". When this bit is set to "0" it indicates that the request is not associated with a particular Local TS, for example in response to the Host sending the *enter_menu()* APDU.

LTS_id: Local TS identifier used by the CICAM to associate the RequestStart object with a local TS when the *LTS_bound_flag* is set to "1". The Host may use the LTS_id value in order to associate the request with a display area where the corresponding program is being displayed, when more than one program are decrypted.

Other Fields: Refer to Table 62 of the DVB-CI specification [2].

6.4.6.3 RequestStartAck APDU

The RequestStartAck() APDU is as defined in TS 101 699 [2], clause 6.5.3. The semantic of the AckCode field is extended as described in table 24.

Table 24: AckCode coding

AckCode	Meaning
0x05	Unattended
	the LTS_id corresponds to a program that is
	currently not attended by the end-user

After the reception of a *RequestStartAck* () APDU with the "Unattended" *AckCode*, the CICAM shall refrain from sending another *RequestStart*() APDU for the same *LTS_id* until either:

- the Host indicates that the program is attended again by use of either the Record Start protocol or Record Stop protocol; or
- the Host indicates that the LTS_id is allocated for another program by use of the ca_pmt() APDU or sd_start() APDU.

6.4.6.4 FileRequest APDU

The FileRequest() APDU is as defined in TS 101 699 [2], clause 6.5.4.

6.4.6.5 FileAck APDU

The FileAck() APDU is as defined in TS 101 699 [2], clause 6.5.5.

6.4.6.6 AppAbortRequest APDU

The AppAbortRequest() APDU is as defined in TS 101 699 [2], clause 6.5.6

6.4.6.7 AppAbortAck APDU

The AppAbortAck() APDU is as defined in TS 101 699 [2], clause 6.5.7

6.4.7 High-Level MMI resource

6.4.7.1 General

In order to support multi-stream functionality, the High-Level MMI resource has a new resource_type defined $(resource_type=2, version=1)$, in which the enq(), menu() and list() APDUs are extended by adding the Local TS identifier (LTS_id) to the APDU syntaxes. The Host may use the additional LTS_id in order to associate the request with a display in situations where the Host is displaying more than one program at a time (e.g. picture-in-picture, mosaic or dual-display function).

The Host and CICAM shall use this extended APDU while multi-stream mode is active.

6.4.7.2 eng APDU

The enq() APDU is extended to include the Local TS identifier as shown in table 25.

Table 25: enq APDU syntax

Syntax	Number of bits	Mnemonic
enq(){		
<pre>enq_tag length field()</pre>	24	uimsbf
reserved	6	bslbf
LTS_bound_flag	1	bslbf
blind_answer	1	bslbf
<pre>if (LTS_bound_flag == 1) {</pre>		
LTS_id }else {	8	uimsbf
reserved	8	bslbf
answer text length	8	uimsbf
for (i=0; i <enq_length -="" 3;="" i++){<="" td=""><td></td><td>diniobi</td></enq_length>		diniobi
text_char	8	uimsbf
}		

LTS_bound_flag: This 1-bit flag indicates that the enquiry object is associated to a particular Local TS when set to "1". When this bit is set to "0" it indicates that the request is not associated with a particular Local TS, for example in response to the Host sending the *enter_menu()* APDU.

LTS_id: Local TS identifier used by the CICAM to associate the enquiry object with a local TS when the *LTS_bound_flag* is set to "1". The Host may use the LTS_id value in order to associate the enquiry object with a display area where the corresponding program is being displayed, when more than one program are decrypted.

Other Fields: Refer to table 47 of EN 50221 [1].

6.4.7.3 answ APDU

The *answ()* APDU is as defined in EN 50221 [1], clause 8.6.5.3. The semantic of the *answ_id* field is extended as described in table 26.

Table 26: answ_id coding

Answ_id	value
Cancel	0x00
Answer	0x01
Reserved	0x02 - 0xFE
Unattended	0xFF

The value "Unattended" indicates that the LTS_id corresponds to a program which is currently not attended by the end-user (for example, the Host is currently recording the program unattended) and the Host cannot display the requested enquiry object.

After the reception of an *answ()* APDU with the "Unattended" *answ_id*, the CICAM shall refrain from sending another High-Level MMI object for the same *LTS_id* until either:

- the Host indicates that the program is attended again by use of either the Record Start Protocol or Record Stop Protocol; or
- the Host indicates that the *LTS_id* is allocated for another program by use of the *ca_pmt()* APDU or the *sd_start()* APDU.

6.4.7.4 menu APDU

The menu() APDU is extended to include the Local TS identifier as shown in table 27.

Table 27: menu APDU syntax

Syntax	Number of bits	Mnemonic
menu(){		
menu tag	24	uimsbf
length field()		
reserved	7	bslbf
LTS bound flag	1 1	bslbf
if (LTS bound flag == 1) {	-	
LTS id	8	uimsbf
<pre>}else {</pre>		diriosi
reserved	8	bslbf
}		50.5.
choice nb	8	uimsbf
TEXT() /* title text */		diriosi
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i <choice !="FF" *="" <="" choice="" i++)="" nb="" nb;="" td="" when="" {=""><td></td><td></td></choice>		
TEXT()		
}		
 }		

LTS_bound_flag: This 1-bit flag indicates that the enquiry object is associated to a particular Local TS when set to "1". When this bit is set to "0" it indicates that the request is not associated with a particular Local TS, for example in response to the Host sending the *enter_menu()* APDU.

LTS_id: Local TS identifier used by the CICAM to associate the menu object with a local TS when the *LTS_bound_flag* is set to "1". The Host may use the LTS_id value in order to associate the menu object with a display area where the corresponding program is being displayed, when more than one program are decrypted.

Other Fields: Refer to Table 49 of EN 50221 [1].

6.4.7.5 menu answ APDU

The menu_answ() APDU is as defined in EN 50221 [1], clause 8.6.5.5. The semantic of the choice_ref field is extended as described in table 28.

Table 28: answ_id coding

choice_ref	value
Cancel	0x00
The number of the choice selected by the user	0x01 - 0xFE
Unattended	0xFF

The value "Unattended" indicates that the LTS_id corresponds to a program which is currently not attended by the user (for example, the Host is currently recording the program unattended) and the Host cannot display the requested menu or list object.

After the reception of a *menu_answ()* APDU with the "Unattended" *answ_id*, the CICAM shall refrain from sending another High-Level MMI object for the same *LTS_id* until either:

- the Host indicates that the program is attended again by use of either the Record Start protocol or Record Stop protocol; or
- the Host indicates that the LTS_id is allocated for another program by use of the *ca_pmt()* APDU or the *sd_start()* APDU.

6.4.7.6 list APDU

The list() APDU is extended to include the Local TS identifier as shown in table 29.

Table 29: list APDU syntax

Syntax	Number of bits	Mnemonic
7.5 mb /) [DITS	
list() {		
list_tag	24	uimsbf
length_field()		
reserved	7	bslbf
LTS_bound_flag	1	bslbf
if (LTS bound flag == 1) {		
LTS id	8	uimsbf
}else {		diriobi
reserved	8	bslbf
}		DSIDI
item nb	8	uimsbf
TEXT() /* title text */	٥	uimsbi
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i <item_nb; !="FF" *="" <="" i++)="" item_nb="" td="" when="" {=""><td></td><td></td></item_nb;>		
TEXT()		
}		
<u> </u>		

LTS_bound_flag: This 1-bit flag indicates that the enquiry object is associated to a particular Local TS when set to "1". When this bit is set to "0" it indicates that the request is not associated with a particular Local TS, for example in response to the Host sending the *enter_menu()* APDU.

LTS_id: Local TS identifier used by the CICAM to associate the list object with a local TS when the *LTS_bound_flag* is set to "1". The Host may use the LTS_id value in order to associate the list object with a display area where the corresponding program is being displayed, when more than one program is being decrypted.

Other Fields: Refer to Table 51 of EN 50221 [1].

6.4.7.7 close mmi APDU

The close_mmi() APDU is as defined in EN 50221 [1], clause 8.6.2.1.

6.4.7.8 display_control APDU

The display_control() APDU is as defined in EN 50221 [1], clause 8.6.2.2.

6.4.7.9 display_reply APDU

The display_reply() APDU is as defined in EN 50221 [1], clause 8.6.2.3.

7 IP delivery Host player mode

7.1 General

The present clause specifies the support of IP-delivered content, which can be either in TS format, ISOBMFF format, or in MPEG DASH format containing one of either TS or ISOBMFF format content.

IP-delivered content can be in either TS or ISOBMFF container format, or either container format embedded in a DASH asset. For any non-TS IP-delivered content the Host shall manage the re-encapsulation of content Samples into a CI PlusTM IP-delivery specific TS-based container format that is applied for the transfer between Host and CICAM, and back to the Host.

In Host Player mode the CICAM may apply CI PlusTM Content Control scrambling for the generated elementary streams to the DRM-protected Elementary Streams that it descrambles.

When carrying IP-delivered content within some non-TS container format, or TS other than conventional CA-protected TS, the TS Interface is said to be operating in Sample Mode, otherwise when carrying a conventional TS it is in Normal Mode.

In Sample Mode, DRM metadata and control information related to the IP-delivered content item being received is passed from the Host to the CICAM via the Command Interface before playback can be started.DRM metadata that changes between Samples, for example the IV, is carried in-band with the Sample data on the TS Interface.

In Sample Mode the CICAM will usually have to buffer incoming data from the Host until it is ready to decrypt Samples. For this reason the generally applicable rule with the decryption of a conventional TS, of constant packet delay through the CICAM, does not apply in Sample Mode.

If date and time information is not available from any broadcast tuner input, the Host and CICAM may acquire date and time information via the IP network interface. The method to acquire date and time information via the IP network interface is out of scope of the present document.

The facility of Host Service Shunning, specified in clause 10 of the CI PlusTM V1.3 specification [3] as applying to broadcast services, is not applicable to services delivered via IP delivery Host player mode.

7.2 TS Interface modes

When the CICAM is used for Sample decryption, the CICAM may buffer a received Sample before it is decrypted and finally sent back through the TS Interface. Buffering of the Sample in the CICAM enables the descrambler used for decrypting that Sample to be set up, typically with a new IV, before the Sample can be decrypted. The CICAM buffering method and the number of descrambling units available in the CICAM is dependent on implementation and not determined by the present document.

The CICAM needs to know when to expect media Samples and when to expect a normal TS. The Host signals this to the CICAM by setting the TS Interface (or a Local TS in the multi-stream context, as described in clause 4.2 and clause 6) into either Sample Mode or Normal Mode.

The configuration of the state of the TS Interface is controlled by the Host using the *sd_start()* and *ca_pmt()*APDUs. The *sd_start()* APDU is a member of the Sample decryption resource, which is specified in clause 7.4.

When the TS Interface changes mode, any data relating to its previous mode in either the Host or the CICAM shall be discarded.

7.3 Command Interface

7.3.1 General

When the TS Interface is configured in Sample Mode, it is necessary that the Host coordinates the delivery of the media file data to the CICAM to enable correct decryption of the content. The CICAM also requires certain information from the metadata in the media file for this purpose.

7.3.2 Playback initiation

The Host controls the configuration of the TS Interface. By default, the TS Interface is in Normal Mode.

The Host shall use the $sd_start()$ APDU to switch the TS Interface, or a Local TS in the multi-stream context, to Sample Mode. On reception of the $sd_start()$ APDU, the CICAM shall terminate the decryption of any pending Normal Mode content previously initiated with the $ca_pmt()$ APDU. When ready, the CICAM shall acknowledge the TS configuration change by sending the $sd_start_reply()$ APDU. From the transmission of the $sd_start()$ APDU, the TS Interface, or a Local TS in the multi-stream context, is considered to be in Sample Mode.

With a single *sd_start()* APDU, the Host may request the decryption of Samples originating from different Tracks, those latter being located in the same media file or in different media files and possibly having their own DRM metadata. Hence, in the rest of the present document, this is referred to as Sample Track decryption.

In the particular case of a TS media file, the Host need not parse the file before it is transmitted on the TS Interface, as it is assumed that the CICAM is capable of extracting the data it needs for decryption. In particular, the Host need not determine the elementary streams contained in the TS before transmission on the TS Interface. It is then considered that a TS media file itself constitutes one Sample Track.

When it is ready to receive Samples, the CICAM shall return an acknowledgement by sending the *sd_start_reply()* APDU. After receiving this acknowledgment, the Host may start sending the first Samples. If the CICAM is not able to decrypt the Samples yet, for example because it is in the process of acquiring the license from a DRM server, this status shall be indicated in the acknowledgement. When the CICAM is finally capable of decrypting the Samples, it shall send a second acknowledgment to the Host, again using the *sd_start_reply()* APDU, with the updated status. Now the CICAM shall start decryption of the Samples and send the decrypted Samples back to the Host on the return TS Interface. The Host shall not send more Sample data than the CICAM buffer can store before receiving the second acknowledgement from the CICAM and confirmation by way of buffer level notification mechanism in the returned TS, specified in clause 7.5.4.3, that the CICAM buffer is being depleted. The CICAM buffer size for the LTS is known from information in the *sd_start_reply()* APDU from the CICAM.

The two-stage acknowledgement mechanism, depicted in figure 5, provides an opportunity to accelerate the presentation of the content and hence improve the user experience.

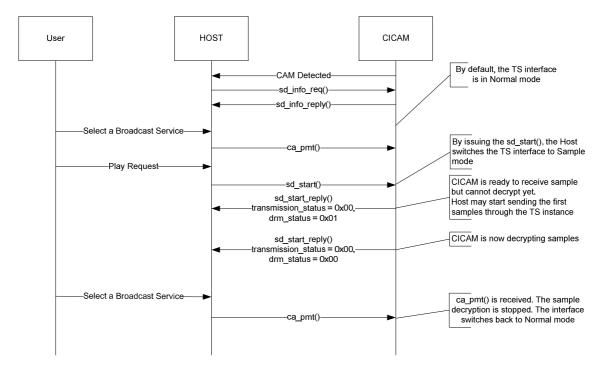


Figure 5: Two-stage playback initiation

If the CICAM is able to determine the final status quickly enough, then only one acknowledgement is sent, as shown in the sequence diagram in figure 6.

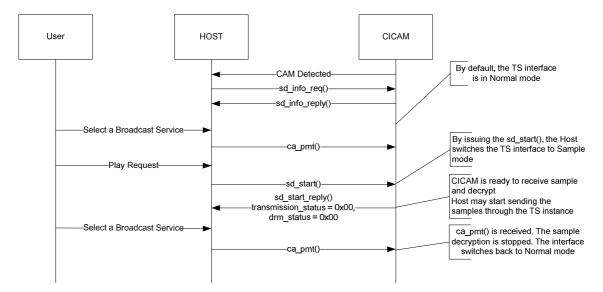


Figure 6: Single-stage playback initiation

7.3.3 Playback execution

From the first *sd_start_reply()* APDU, the Host may start transmitting the Samples over the TS Interface. The resulting TS is compliant with the structure defined in clause 7.5. The Host shall respect the CICAM Buffering constraints, as defined in clause 7.5.4.

During playback execution, the Host may update the list of Sample Tracks involved. For instance, the user may change the audio language, which may result in a change of the audio Track in use. Such changes of Track shall be indicated by the Host using the *sd_update()* APDU, as shown in figure 7.

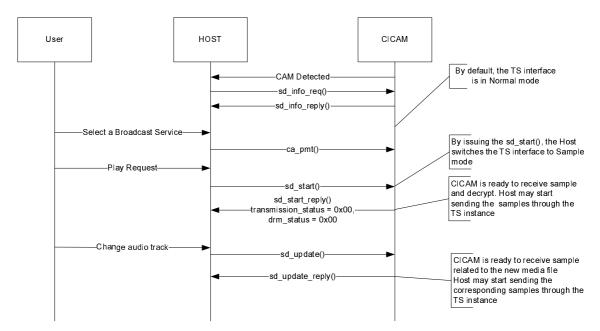


Figure 7: Playback execution sequence diagram

During playback, the Host may provide an update of the DRM metadata associated to the list of Sample Tracks by using the *sd_update()* APDU.

During playback, the Host may change completely the list of Sample Tracks involved by issuing another $sd_start()$ APDU, typically when the user has selected another content item that requires operation in Sample Mode. Upon reception of an $sd_start()$ APDU when already in Sample Mode, the CICAM shall terminate any decryption in progress, shall flush its buffers and shall prepare for the decryption of the Samples corresponding to the new list of files. The CICAM shall send one or more acknowledgments as described previously. This re-start process is shown in the sequence diagram in figure 8.

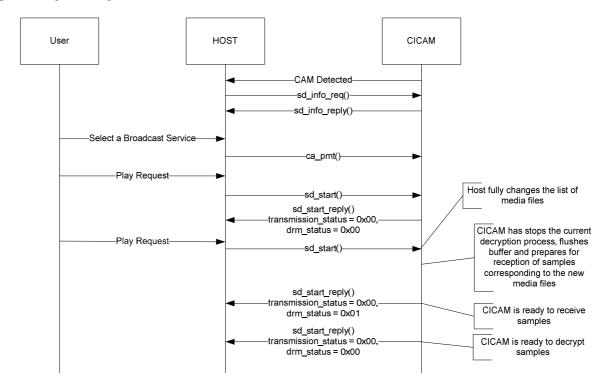


Figure 8: Playback re-start sequence diagram

7.3.4 Playback termination

When the Host has finished sending the media Samples to the CICAM, for instance because the Host has reached the end of the media file, or because the user wishes to stop viewing it, the Host may send the $ca_pmt()$ APDU. This puts the TS Interface, or a Local TS in the multi-stream context, back into Normal Mode, ready to start the decryption of a broadcast stream.

7.4 Sample decryption resource

7.4.1 Resource usage

The Sample decryption resource is defined to control the decryption by the CICAM of a set of consecutive media Samples packaged into a TS that is compliant with the MPEG-2 Systems specification [4].

The Host may send an $sd_info_req()$ APDU in order to request the list of DRM systems embedded in the CICAM that support Sample decryption. The CICAM shall respond with a list of supported drm_system_id and Content Protection System UUID by using the $sd_info_reply()$ APDU.

The Host shall declare one or more Sample Tracks for which Samples are transported over the TS Interface for decryption in the CICAM by using the $sd_start()$ APDU. The CICAM shall respond with a first $sd_start_reply()$ APDU when it is ready to receive Samples and a second $sd_start_reply()$ APDU when it is ready to decrypt the Samples. If the CICAM is fast enough, then only one $sd_start_reply()$ APDU may be returned by the CICAM, containing both of the relevant updated status fields.

The Host may update the list of the Sample Tracks involved in the Sample decryption process by use of the *sd_update()* APDU. The Host may add one or more new Sample Tracks to the list, or may remove one or more Sample Tracks that are no longer in use. The CICAM shall acknowledge the update using the *sd_update_reply()* APDU.

The Host may update the DRM Metadata associated with the existing Sample Tracks with the sd_update() APDU.

A Sample Track is identified by the *track_PID* allocated by the Host in the *sd_start()* APDU.

Table 30 provides a summary of the attributes of the Sample decryption resource.

Table 30: Sample decryption resource summary

	Resou	rce			Application Ob	ject	Direction
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host CICA M
Sample	00920041	146	1	1	sd_info_req	9F 98 00	\rightarrow
Decryption					sd_info_reply	9F 98 01	+
					sd_start	9F 98 02	\rightarrow
					sd_start_reply	9F 98 03	+
					sd_update	9F 98 04	\rightarrow
					sd_update_reply	9F 98 05	+

The remainder of clause 7.4 specifies the APDUs defined for the Sample decryption resource.

7.4.2 sd_info_req APDU

The Host shall send this APDU to the CICAM to determine the list of the DRM systems embedded on the CICAM that support the Sample decryption feature. The CICAM shall reply with an *sd_info_reply()* APDU.

Table 31: sd_info_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>sd_info_req() { sd_info_req_tag length_field() }</pre>	24	uimsbf

The fields are defined as follows:

sd info req tag: This 24 bit field with value 0x9F9800 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

7.4.3 sd_info_reply APDU

The CICAM shall send this APDU to the Host in response to the $sd_info_req()$ APDU. It lists the drm_system_id and UUID for each content protection system or DRM system that the CICAM supports for Sample decryption. If a DRM system can be identified by either its drm_system_id or its UUID then both shall be communicated.

Table 32: sd_info_reply APDU Syntax

Syntax	Number of bits	Mnemonic
<pre>sd_info_reply() {</pre>		
sd_info_reply_tag	24	uimsbf
length_field()	16	
number_of_drm_system_id	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm_system_id	16	uimsbf
}		
number_of_drm_uuid	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm uuid	128	uimsbf
} _		
]}		

sd info reply tag: This 24 bit field with value 0x9F9801 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

number_of_drm_system_id: The number of entries contained in the following list of DRM system identifiers. This list shall not contain the identifiers of the broadcast CA Systems that may also be supported by the CICAM, unless the same CA/DRM system supports both broadcast CA decryption and Sample decryption.

NOTE: The list of supported broadcast CA Systems is provided by the *CA_info_reply()* APDU in response to the *CA_info()* APDU, as specified in EN50221 [1] and CI PlusTM V1.3 [3] clause E.17.3.**drm_system_id:**The identification of a DRM System implemented by the CICAM for which the Sample decryption is supported. Values for *drm_system_id* are the same as values for *ca_system_id* as defined in the allocation of identifiers and codes for DVB systems [11].

number_of_drm_uuid: The number of entries contained in the following list of DRM UUIDs.

drm_uuid: UUID of DRM system supported by the CICAM.

7.4.4 sd_start APDU

The Host shall send the *sd_start()* APDU to the CICAM to indicate the start of Sample decryption for one or more Tracks, and that the TS Interface, or a Local TS in the multi-stream context, is now in Sample Mode. Before sending this APDU, the Host shall stop sending any TS packets relating to any previous activity on this Local TS (in the multi-stream case) (e.g. TS packets from a broadcast tuner).

The Host shall declare the *program_number* to be used by the CICAM in the URI messages.

For each Track the Host shall indicate the PID (*track_PID*) on which the corresponding Samples will be transmitted. The *track_PID* is then used as the identifier of the Track in any subsequent *sd_update()* APDUs. The Host shall ensure that the PIDs signalled in this APDU are all unique and not used for any other purpose. The values from 0x0000 to 0x001F are reserved.

If the Host has identified some metadata associated with a DRM system supported by the CICAM, then it shall add a DRM metadata loop for each Track in the *sd_start()* APDU. If any DRM metadata applies to all Tracks, then the Host shall repeat it for each Track.

Table 33: sd_start APDU syntax

Syntax	Number of bits	Mnemonic
sd_start() {		
sd_start_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
program_number	16	uimsbf
reserved	7	bslbf
ts_flag	1	uimsbf
if (ts_flag == 1) {		
number_of_metadata_records	8	uimsbf
for (i=0; i <number_of_metadata_records; i++)="" td="" {<=""><td></td><td></td></number_of_metadata_records;>		
drm_metadata_source	8	uimsbf
drm_system_id	16	uimsbf
drm_uuid	128	uimsbf
drm_metadata_length	16	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm_metadata_byte	8	uimsbf
}		
}		
}		
if (ts_flag == 0){		
number_of_sample_tracks	8	uimsbf
for (i=0;i <number_of_sample_tracks;i++){< td=""><td></td><td></td></number_of_sample_tracks;i++){<>		
reserved	3	bslbf
track_PID	13	uimsbf
number_of_metadata_records	8	uimsbf
for (i=0; i <number_of_metadata_records; i++)="" td="" {<=""><td></td><td></td></number_of_metadata_records;>		
drm_metadata_source	8	uimsbf
drm_system_id	16	uimsbf
drm_uuid	128	uimsbf
drm_metadata_length	16	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm_metadata_byte	8	uimsbf
}		
}		
}		
\ \		
[}		

sd_start_tag: This 24 bit field with value 0x9F9802 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_id: The identifier of the Local TS.

program_number: The *program_number* to be used by the CICAM in the URI messages. For a TS this need not to be the same value as the *program_number* field in the PMT of incoming TS. For non-TS content *program_number* might not be applicable with the format of incoming content and the Host may set *program_number* to any value.

ts_flag: Indicates whether the Sample decryption request is for a TS (value 0b1) or non-TS format (value 0b0).

number_of_Sample_Tracks: The number of Sample Tracks for which Samples shall be decrypted.

track_PID: The PID on which the Samples for the described Sample Track are sent by the Host.

number_of_metadata_records: The number of DRM metadata records in the following loop.

drm_metadata_source: The source of the metadata as defined in table 34.

Table 34: DRM metadata source

DRM Metadata source	Value
Undefined	0x00
Content Access Streaming Descriptor (CASD) - DRM Generic Data.	0x01
The DRMGenericData element of the DRMControlInformation element of the Content	
Access Streaming Descriptor is copied into the drm_metadata_byte array as a UTF-8	
encoded string.	
The Content Access Streeming Descriptor is defined in continue 4.7 and 5.2 of the	
The Content Access Streaming Descriptor is defined in sections 4.7 and E.2 of the OIPF Declarative Application Environment specification [12].	
The DRMControllnformation element is defined in section 3.3.2 of the OIPF Content	
Metadata specification [13].	
Content Access Streaming Descriptor (CASD) - DRM Private Data.	0x02
The DRMPrivateData element of the DRMControllnformation element of the Content	OXOZ
Access Streaming Descriptor is copied into the drm_metadata_byte array as a UTF-8	
encoded string.	
The Content Access Streaming Descriptor is defined in sections 4.7 and E.2 of the	
OIPF Declarative Application Environment specification [12].	
The DRMControlInformation element is defined in section 3.3.2 of the OIPF Content	
Metadata specification [13].	
Common Encryption (CENC) - Protection System Specific Header ('pssh') box.	0x03
The Data field of the 'pssh' box is copied into the drm_metadata_byte array.	
The 'pssh' box is defined in the MPEG Common Encryption specification [14].	0.04
Media Presentation Description (MPD) - Content Protection Element. The ContentProtection element from the Representation element for the Track to be	0x04
decrypted is copied into the drm_metadata_byte array as a UTF-8 encoded string.	
decrypted is copied into the diff_filetadata_byte array as a 0 ft -0 efficuled string.	
The Media Presentation Description is defined in the MPEG DASH specification [15].	
The ContentProtection element is defined in sections 5.8.4.1 and 5.8.5.2 of that	
document.	
ISOBMFF - Protection Scheme Information ('sinf') box.	0x05
The Data field of the 'sinf' box is copied into the drm_metadata_byte array. The 'sinf'	
box is defined in the ISO Base Media File Format specification [8].	
Online SDT (OSDT) - DRM Generic Data.	0x06
The DRMGenericData element of the DRMControlInformation element of the OSDT is	
copied into the drm_metadata_byte array as a UTF-8 encoded string.	
The OSDT is defined in Annex D of the present document.	
The DRMControlInformation element is defined in section 3.3.2 of the OIPF Content	
Metadata specification [13].	
Online SDT (OSDT) - DRM Private Data.	0x07
The DRMPrivateData element of the DRMControlInformation element of the OSDT is	
copied into the drm_metadata_byte array as a UTF-8 encoded string.	
The OSDT is defined in Annex D of the present document.	
The DRMControlInformation element is defined in section 3.3.2 of the OIPF Content	
Metadata specification [13].	
Reserved	0x08-0xFF

drm_system_id: The identification of the DRM system to which the metadata is related. Values for *drm_system_id* are the same as values for *ca_system_id* as defined in the allocation of identifiers and codes for DVB systems [11]. If *drm_system_id* is not used for identification of the DRM then this field shall be set to 0xFFFF.

drm_uuid: The UUID of the DRM system to which the metadata is related. If *drm_uuid* is not used for identification of the DRM, all bytes of this field shall be set to 0xFF.

drm_metadata_length: The length in bytes of the following DRM metadata.

drm_metadata_byte: The DRM metadata, as defined by table 34.

7.4.5 sd_start_reply APDU

The CICAM shall send the *sd_start_reply()* APDU to the Host as a reply to the *sd_start()* APDU when it is ready to receive the first Sample to be decrypted.

The *sd_start_reply()* APDU informs the Host of:

- The CICAM's readiness to receive transport packets over the TS Interface (transmission_status).
- The CICAM's capability to decrypt the encrypted transport packets (*drm_status*).
- The size of the buffer (buffer_size) allocated by the CICAM for all declared Sample Tracks in the sd_start() APDU. The minimum returned buffer_size shall be 5 000 TS packets.

The CICAM may send more than one $sd_start_reply()$ after the reception of the $sd_start()$ APDU. For example, the CICAM may send a first $sd_start_reply()$ with $transmission_status$ set to 0x00 (ready to receive) and drm_status set to 0x01 (status currently undetermined), and then a second $sd_start_reply()$ with $transmission_status$ set to 0x00 (ready to receive) and drm_status set to 0x00 (decryption possible). This is to allow the Host to start filling the CICAM buffer from the reception of the first $sd_start_reply()$, while the CICAM is preparing itself to decrypt the Samples, and hence improving the user experience by accelerating the presentation of content.

The $buffer_size$ value set in the first $sd_start_reply()$ shall not be changed by the CICAM in a subsequent $sd_start_reply()$.

Syntax Number **Mnemonic** of bits sd_start_reply() sd start_tag uimsbf 2.4 length field() LTS id uimsbf g transmission_status 8 uimsbf drm status 8 uimsbf drm system id 16 uimsbf drm uuid 16*8 uimbsf buffer size 16 uimbsf

Table 35: sd_start_reply APDU syntax

The fields are defined as follows:

sd_start_reply_tag: This 24 bit field with value 0x9F9803 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_id: The identifier of the Local TS.

transmission_status: This field contains the transmission status of the CICAM. A value of 0x00 (ready to receive) indicates that the CICAM is ready to receive Samples to be decrypted. Other values indicate that the CICAM is not ready to receive. Table 36 lists the possible values.

Table 36: Possible values for transmission_status

transmission_status	Value
Ready to receive	0x00
Error - CICAM busy	0x01
Error - other reason	0x02
Reserved	0x03-0xFF

drm_status: This byte returns the DRM status of the CICAM. Table 37 lists the possible values.

Table 37: Possible values for drm_status

drm_status	Value
Decryption possible	0x00
Status currently undetermined	0x01
Error - no entitlement	0x02
Reserved	0x03-0xFF

drm_system_id: The identification of the DRM system that the CICAM will use to decrypt the Samples. Values for *drm_system_id* are the same as values for *ca_system_id* as defined in the allocation of identifiers and codes for DVB systems [11]. If *drm_system_id* is not used for identification of the DRM, this field shall be set to 0xFFFF.

drm_uuid: The UUID of the DRM which the CICAM will use to decrypt the Samples. If drm_uuid_id is not used for identification of the DRM, all bytes of this field shall be set to 0xFF.

buffer_size: The CICAM buffer size allocated for the decryption of the Sample Tracks, expressed in number of transport packets. If more than one Sample Track was declared in the *sd_start()* APDU, the buffer will be shared between them.

7.4.6 sd_update APDU

The Host shall send the *sd_update()* APDU to the CICAM for any of the following reasons:

- To indicate a change in the list of Sample Tracks for which Samples shall be decrypted. Typically, this APDU
 may be sent following a change in the user selection resulting in a change in the set of Sample Tracks to be
 decrypted.
- To provide some additional DRM metadata that may be necessary for the CICAM to continue Sample decryption. Typically, this APDU may be sent before a key rotation in order to provide the necessary metadata to the CICAM to compute the applicable keys due to the key rotation.

For non-TS content, each Sample Track is referenced by its *track_PID*, as declared in a previous *sd_start()* or *sd_update()* APDU:

- If a *track_PID* is still listed, the CICAM shall continue the decryption of the corresponding Samples.
- If a *track_PID* is no longer listed, the CICAM shall stop the decryption of corresponding Samples and flush any TS packets with this PID from its buffer. The Host shall stop sending any TS packets with this PID before sending this APDU.
- All sample tracks to be decrypted shall be listed, even if they were notified to the CICAM in previous APDUs; however, in this case, the metadata need not be repeated if it has not changed.
- If a new *track_PID* is listed, the CICAM shall prepare itself for decryption and respond with a *sd_update_reply()* when ready.

For TS content there is a single Sample Track, hence there is no possibility to add or remove a Track. As a consequence, there is no need for a Track identifier, and the $sd_update()$ APDU may be used only to provide additional DRM metadata.

Table 38: sd_update APDU syntax

Syntax	Number of bits	Mnemonic
sd update() {		
sd update tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
reserved	7	bslbf
ts_flag	1	bslbf
if (ts_flag == 1) {		
number_of_metadata_records	8	uimsbf
<pre>for (i=0; i<number_of_metadata_records; i++)="" pre="" {<=""></number_of_metadata_records;></pre>		
drm_metadata_source	8	uimsbf
drm_system_id	16	uimsbf
drm_uuid	128	uimsbf
drm_metadata_length	16	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm_metadata_byte	8	uimsbf
}		
}		
}		
if (ts_flag == 0) {		
number_of_Sample Tracks	8	uimsbf
<pre>for (i=0; i<number_of_sample_tracks; i++)="" pre="" {<=""></number_of_sample_tracks;></pre>		
reserved	3	bslbf
Sample_track_PID	13	uimsbf
number_of_metadata_records	8	uimsbf
<pre>for (i=0; i<number_of_metadata_records; i++)="" pre="" {<=""></number_of_metadata_records;></pre>		
drm_metadata_source	8	uimsbf
drm_system_id	16	uimsbf
drm_uuid	128	uimsbf
drm_metadata_length	16	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
drm_metadata_byte	8	uimsbf
}		
}		
}		
}		
}		

sd_update_tag: This 24 bit field with value 0x9F9804 identifies this APDU

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_id: The identifier of the Local TS for which the update applies.

ts_flag: The ts_flag is a 1-bit field indicating that the Sample decryption request is related to a TS Sample Track. Set to zero otherwise. The ts_flag shall have the same value as in the *sd_start()* APDU.

number_of_Sample Tracks: The number of Sample Tracks for which Samples shall be decrypted.

Sample_track_PID: The PID on which the Samples for the described Sample Track are available.

number_of_metadata_records: The number of DRM metadata records in the following loop.

drm_metadata_source: The source of the metadata according to the table 21.

drm_system_id: The identification of the DRM to which the metadata is related. Values for drm_system_id are values for ca_system_id as defined in TS 101 162 [11]. If drm_system_id is not used for identification of the DRM, this field shall be set to 0xFFFF.

drm_uuid: The UUID of the DRM to which the metadata is related. If drm_uuid_id is not used for identification of the DRM, all bytes of this filed shall be set to 0xFF.

drm_metadata_length: The length in bytes of the following DRM metadata.

drm_metadata_byte: The DRM metadata, as defined in table 34.

7.4.7 sd_update_reply APDU

The CICAM shall send the *sd_update_reply()* APDU to the Host as a reply to the *sd_update()* APDU.

If the *sd_update()* adds one or more Sample Tracks, then the Host may start transmitting TS packets containing Samples corresponding to the new Sample Tracks after the *sd_update_reply()* is returned by the CICAM.

If the $sd_update()$ provides some additional DRM metadata, then the Host may start transmitting TS packets containing Samples encrypting with the additional DRM metadata after the $sd_update_reply()$ is returned by the CICAM.

Table 39: sd_update_reply APDU syntax

Syntax	Number of bits	Mnemonic
<pre>sd_update_reply() { sd_update_reply_tag length field()</pre>	24	uimsbf
LTS_id drm status	8	uimsbf uimsbf
}		dIMSDI

The fields are defined as follows:

sd_update_reply_tag: This 24 bit field with value 0x9F9805 identifies this APDU.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS id: The identifier of the Local TS.

drm_status: This byte returns the DRM status of the CICAM. Table 37 lists the possible values.

7.5 TS Interface

7.5.1 General

If the Sample Track is a TS, it is straightforward and efficient to send the TS packets directly through the TS Interface without additional encapsulation. There are, however, additional signalling facilities defined to facilitate the TS transfer in *Sample Mode*. These are specified in clause 7.5.2.

If the Sample Track is of a different type than TS, then additional signalling and encapsulation for the Samples shall be applied as specified in clause 7.5.3.

7.5.2 TS content carriage

7.5.2.1 Host output

When the type of the Sample Track is TS, the Host shall send the TS packets from the original TS Sample Track to the CICAM over the TS Interface.

The Host need not parse and shall not make any assumptions about the TS Sample Track before it is sent over the TS Interface. The CICAM shall be capable of analysing the received TS, typically extracting the PSI, identifying the PIDs carrying ECM (if any), and identifying the PIDs to be decrypted.

The Host may send Flush Tables (FLT) on the comms PID.

The Host may send one or more Sample End Tables (SET) on the same PIDs that are used to carry the Sample Tracks, using the method specified in clause 7.5.5.3. SET shall not be sent on the *comms_PID*, since then there would be no guarantee of the SET arriving in the correct sequence with respect to the final Track Sample data.

The Host shall not send a Sample Start Table (SST).

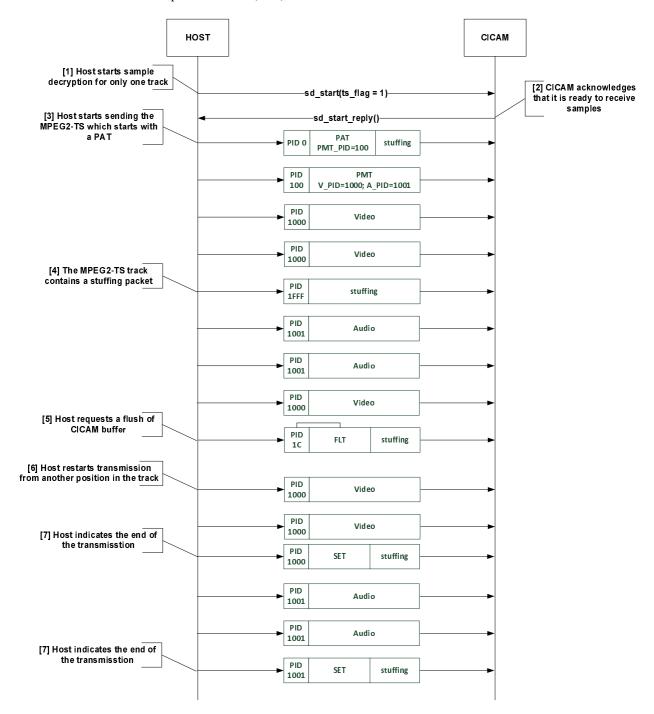


Figure 9: Example sequence with TS content carriage from Host to CICAM

NOTE: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending an *sd_start()* APDU. There is only one Track declared, which is in TS format (ts_flag = 1).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with an sd_start_reply() APDU.
- 3) The Host starts the transmission of the TS Track. Very likely, it starts with a PAT and a PMT, but the Host need not make assumptions on the TS Track content and is not intended to make any check or manipulation of the TS Track.
- 4) At some point, the TS Track contains a stuffing packet.

- 5) The Host requests the CICAM to flush its buffer by sending a FLT on the comms_PID. Very likely, the Host is seeking into the Track.
- 6) The Host restarts the transmission from another position in the TS Track. The Host need not wait for receiving back the FLT from the CICAM.
- 7) The Host indicates the end of Track transmission by sending a SET on the Sample Track PID.

7.5.2.2 CICAM output

For the decryption of a TS Sample Track, the CICAM shall output the transport packets in the same order as they were received from the Host.

The CICAM may buffer the received transport packets as described in clause 4.3.

The CICAM shall inform the Host continually about the CICAM buffer fullness level as specified in clause 7.5.4.3.

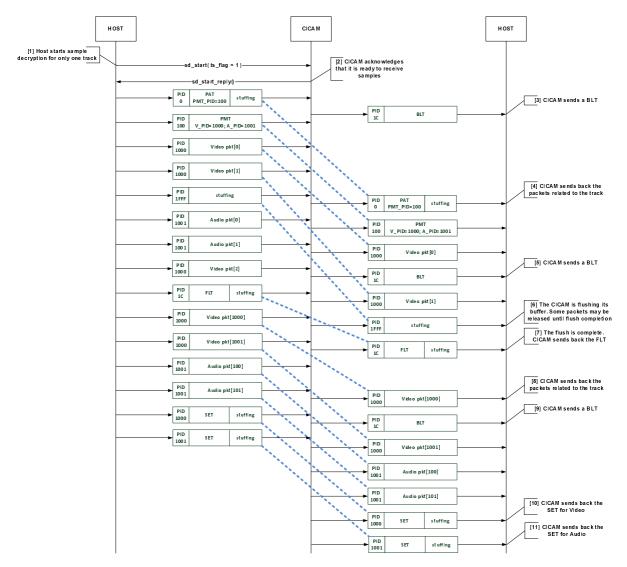


Figure 10: Example sequence with TS content carriage from Host to CICAM and back

NOTE: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending an $sd_start()$ APDU. There is only one Track declared, which is TS $(ts_flag = 1)$.
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with the *sd_start_reply()* APDU.
- 3) The CICAM sends the first BLT on the *comms PID*. The buffer level indicated is the maximum.

- 4) The CICAM starts sending back the packets related to the Track. Packets originally in the clear are sent back in the clear on the TS Interface. Packets originally encrypted are decrypted and optionally re-encrypted (depending on the EMI value) before they are sent back on the TS Interface.
- 5) The CICAM sends a BLT on the *comms_PID*.
- 6) The CICAM has received the FLT from the Host. The CICAM flushes its buffer. Until completion, some packets received before the flush may be released on the TS Interface.
- 7) The CICAM has completed the flush and sends back the FLT unmodified as an acknowledgement.
- 8) The CICAM sends back the packets related to the Track over the TS Interface.
- 9) The CICAM sends a BLT on the *comms_PID*.
- 10) The CICAM returns the SET for video.
- 11) The CICAM returns the SET for audio.

7.5.2.3 Multiple TS Sample Tracks

In order to avoid PID collisions on the Local TS configured in Sample Mode:

- The Local TS shall contain a maximum of one TS Sample Track.
- The Local TS shall not contain both a TS Sample Track and a non-TS Sample Track.

Concurrent descrambling may be possible using multiple Local TSs as specified in clause 6 on multi-stream handling, depending upon supported features and resource availability.

7.5.3 Non-TS content carriage

7.5.3.1 Host output

7.5.3.1.1 General

When the TS Interface is configured in Sample Mode, and the Sample Track(s) are not TS, the TS sent by the Host to the CICAM shall contain TS packets only with the PIDs declared in the *sd_start()* or *sd_update()* APDUs.

For each Sample Track, the TS sent by the Host over the TS Interface shall contain a sequence of Samples pertaining to the said Sample Track in the assigned PID.

7.5.3.1.2 Transmitting Samples

Before the transmission of a Sample, the Host shall announce that Sample by sending a Sample Start Table (SST) in the TS using the same PID as the Sample Track. A Host may repeat the SST provided that the last repetition of the SST precedes the start of the Sample. The SST contains the credentials necessary for the decryption of the Sample, typically the initialisation vector (IV) and the key identifier.

The SST contains a *tsc_parity_bit*, which indicates the parity of the *transport_scrambling_control* bits of the transport packets containing encrypted data of the announced Sample. The Host shall toggle the *tsc_parity_bit* in the SST of the next Sample of a Sample Track.

After the SST for a Sample has been transmitted, the Host may start the transmission of the Sample. Unless some error condition occurs, the Host shall complete the transmission of the previous Sample by sending a SET before sending the SST section of the next Sample for the same Track.

After the transmission of the transport packet containing the end of a Sample, the Host shall send a Sample End Table (SET) in the TS on the same PID as the Sample Track (*track_PID*) using the method specified in clause 7.5.5.3. The Host shall complete the transmission of a Sample before sending a SET for that Sample.

The sequence diagram in figure 11 illustrates informatively the usage of the SST and SET by the Host.

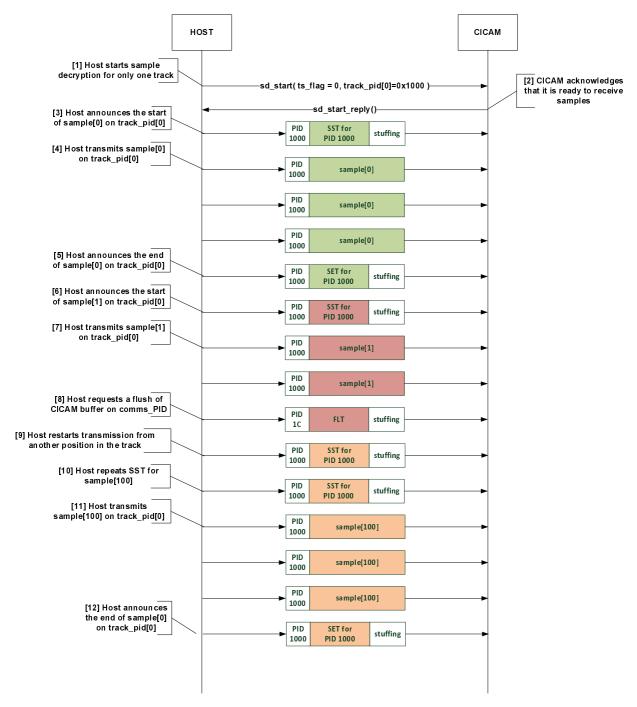


Figure 11: Example sequence with non-TS content carriage from Host to CICAM

NOTE: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending a *sd_start()* APDU. There is only one Track declared (Track[0] on PID 0x1000).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with an *sd_start_reply()* APDU.
- 3) Host announces the start of the Sample[0] by sending a SST on the *track_PID*.
- 4) Host transmits the Sample[0] on track_PID[0]=0x1000.
- 5) Transmission of Sample[0] is complete. The Host announces the end of the Sample[0] by sending a SET on the *track_PID*.
- 6) The Host announces the start of the Sample[1] by sending a SST on the *track_PID*.

- 7) Host transmits the first part of the Sample[1] on track_PID[1]=0x1000.
- 8) Host requests the CICAM to flush its buffer by sending a FLT on the *comms_PID*.
- 9) The Host restarts the transmission from another position in the TS Track. The Host does not wait for receiving back the FLT from the CICAM. Host announces the start of the Sample[100] by sending a SST on *track_PID*. The Host is very likely performing a jump operation.
- 10) Host repeat the SST for Sample[100].
- 11) Host transmits the Sample[100] on track_PID[0]=0x1000.
- 12) Transmission of Sample[100] is complete. The Host announces the end of the Sample[100] by sending a SET on the *track_PID*.

At any time, the Host may request the CICAM to flush its Sample buffer by sending a Flush Table (FLT) in the TS using the *comms_PID* value of 0x001C, as designated for link-local in-band signalling in the DVB SI specification [10] clause 5.1.3. The Host shall stop the transmission of the pending Sample(s) before sending the FLT. Immediately after the transmission of the FLT, the Host may start the transmission of some new Samples, announced with SST.

7.5.3.1.3 Managing multiple Tracks

When more than one Track is declared, the Host may interleave Samples originating from different Tracks, as illustrated in the sequence diagram in figure 12.

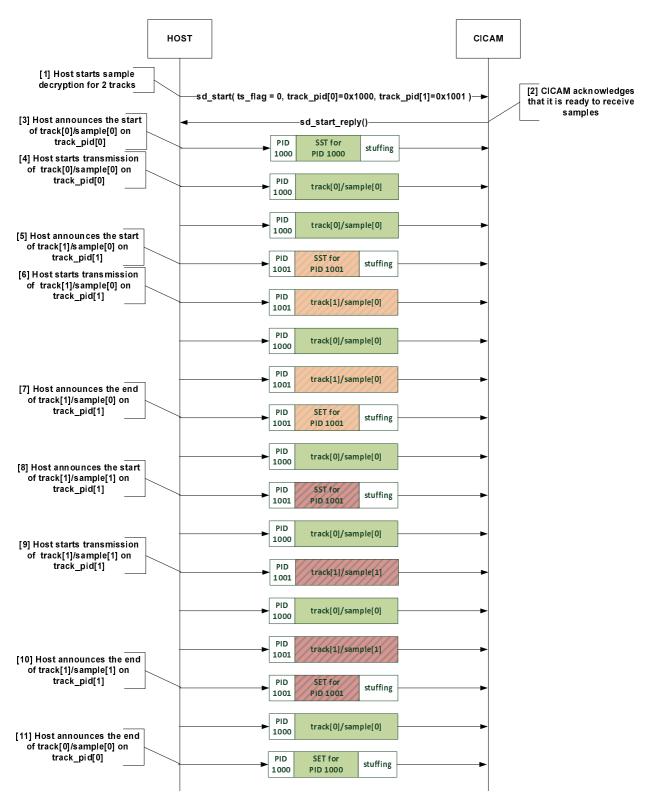


Figure 12: Example sequence with multiple-TS content carriage from Host to CICAM

NOTE: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending an $sd_start()$ APDU. There are 2 Tracks declared (Track[0] on PID 0x1000 and Track[1] on PID 0x1001), which are not TS (ts_flag=0).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with an sd_start_reply() APDU.
- 3) Host announces the start of the Track[0]/Sample[0] by sending a SST on the *track_PID*.

- 4) Host starts transmission of the Track[0]/Sample[0] on track_PID[0]=0x1000.
- 5) Host announces the start of the Track[1]/Sample[0] by sending a SST on *track_PID*.
- 6) Host starts transmission of the Track[1]/Sample[0] on track_PID[0] = 0x1001. The 2 Samples are now interleaved.
- 7) Transmission of Track[1]/Sample[0] is complete. The Host announces the end by sending a SET on the *track PID*.
- 8) Host announces the start of the Track[1]/Sample[1] by sending a SST on the *track_PID*. Track[0]/Sample[0] is still transmitted.
- 9) Host starts transmission of the Track[1]/Sample[1] on track_PID[0] = 0x1001. The 2 Samples are now interleaved.
- 10) Transmission of Track[1]/Sample[1] is complete. Track[0]/Sample[0] is still transmitted.
- 11) Transmission of Track[0]/Sample[0] is complete.

7.5.3.1.4 Track list update

The Host may update at any time the list of the pending Sample Tracks by sending an $sd_update()$ APDU. If a Track is added, the Host shall not start sending Samples related to this additional Track until an affirmative $sd_update_reply()$ is received from the CICAM. If a previously declared Track is removed, the Host shall stop sending Samples related to the removed Track before sending the $sd_update()$ APDU. The sequence diagram in figure 13 illustrates the Host output when updating the Track list.

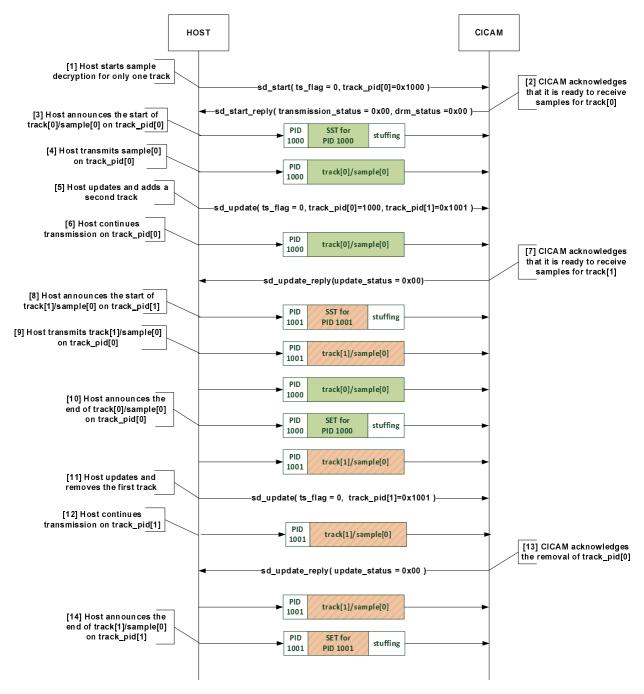


Figure 13: Example sequence with non-TS content Track list update

NOTE: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending a *sd_start()* APDU. There is one Track declared (Track[0] on PID 0x1000), which are not TS (*ts_flag* = 0).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with the *sd_start_reply()* APDU.
- 3) Host announces the start of the Track[0]/Sample[0] by sending a SST on the *track_PID*.
- 4) Host transmits the Track[0]/Sample[0] on $track_PID[0] = 0x1000$.
- 5) Host updates the Track lists by sending an *sd_update()* APDU. The Track list now contains 2 Tracks (Track[0] on PID 0x1000 and Track[1] on PID 0x1001).
- 6) Host continues the transmission of Sample for the Track[0].

- 7) The CICAM acknowledges that it is now ready to decrypt Samples from all Tracks, including the added Track[1].
- 8) Host announces the start of the Track[1]/Sample[0] by sending a SST on *track_PID*.
- 9) Host transmits the Track[1]/Sample[0] on $track_PID[1] = 0x1001$.
- 10) Transmission of Track[0]/Sample[0] is complete. The Host announces the end of the Sample by sending a SET on the *track PID*.
- 11) Host updates the Track list by sending an *sd_update()* APDU. The Track list now contains only one Track (Track[1] on PID 0x1001).
- 12) Host continues the transmission of Sample for the Track[1].
- 13) The CICAM acknowledges the removal of the Track[0]. The CICAM may have de-allocated some descrambling units.
- 14) Transmission of Track[1]/Sample[0] is complete. The Host announces the end of the Sample by sending a SET on the *track_PID*.

7.5.3.2 TS packet headers

For a packet transferred from the Host to the CICAM and containing data from a Sample, the TS packet header bits shall be set as follows:

- The sync byte shall be set to the *LTS_id* of the Local TS.
- The *transport_error_indicator* shall be set to 0b0.
- The *payload_unit_start_indicator* of a packet containing the start of a Sample shall be set to 0b1. Otherwise, the *payload_unit_start_indicator* shall be set to 0b0.
- The *pointer_field* shall be set to 0 when the *payload_unit_start_indicator* is set to 0b1, otherwise it shall not be present.
- The *transport priority* shall be set to 0b0.
- The PID shall be the track PID of the corresponding Track, as defined in the sd start() or sd update() APDU.
- The *transport_scrambling_control* shall be set to either 0b10 or 0b11 for a packet containing encrypted data from a Sample. For a given Sample, the Host shall use the *transport_scrambling_control* value according to the *tsc_parity_bit* indicated in the SST section of the corresponding sample, as defined in clause 7.5.5.4, for all of the packets containing encrypted data of the said sample. The *transport_scrambling_control* field shall be set to 0b00 for a packet containing clear data from a sample.
- The adaptation field control shall be set to:
 - either 0b01 when the TS packet contains only Sample data; or
 - 0b11 when the TS packet contains an adaptation field for the purpose of padding, followed by Sample data.
- The *continuity_counter* shall be as defined in MPEG-2 Systems [4]. The *continuity_counter* may be reset for the packet containing the start of a Sample.

A TS packet payload shall only contain data from one Sample.

The Host shall use the *adaptation_field()* only for padding purpose. When *adaptation_field()* is used, it shall contain only *stuffing_byte*, and all the indicators and flags in the second byte of the *adaptation_field()* shall be set to zero.

The Host shall restrict the usage of the adaptation_field() to:

• a TS packet that contains the end of a Sample;

- a TS packet that contains clear data when the next transport packet on the same track_PID contains encrypted data:
- a TS packet that contains encrypted data when the next transport packet on the same track_PID contains clear data.

7.5.3.3 CICAM output

For a given Sample Track, the CICAM shall output the transport packets with the *track_PID* in the same order that they were received. The structure of the transport packets shall not be changed except that:

- The CICAM may have decrypted the payload of some of the transport packets.
- The CICAM may have re-encrypted some of the decrypted transport packets; the *transport-scrambling_control* bits of these packets shall be set according to the CI PlusTM V1.3 specification [3], in clause 5.6.1.
- The CICAM may use a different starting value of *continuity_counter*. However, the CICAM shall use a compliant sequence of *continuity_counter* values within the packets of a Sample.

The relative order of transport packets for different Sample Tracks may be different on output from the CICAM to the order sent by the Host, as illustrated in figure 14.

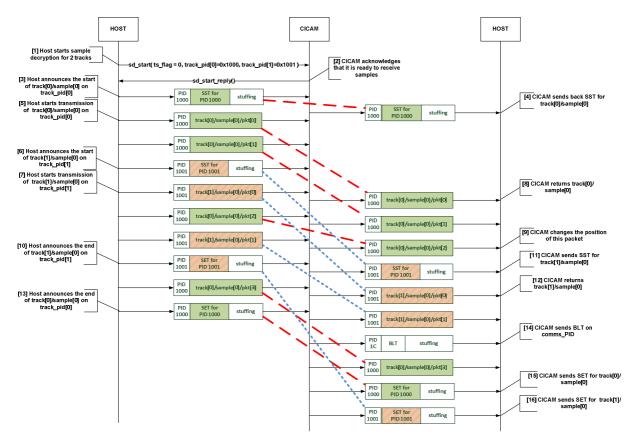


Figure 14: Example sequence with multiple non-TS content streams

NOTE 1: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending an *sd_start()* APDU. There are 2 Tracks declared (Track[0] on PID 0x1000 and Track[1] on PID 0x1001), which are not TS (ts_flag=0).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with the *sd_start_reply()* APDU.
- 3) Host announces the start of the Track[0]/Sample[0] by sending a SST on track_PID[0].
- 4) The CICAM returns SST of Track[0]/Sample[0].

- 5) Host starts the transmission of Track[0]/Sample[0].
- 6) Host announces the start of the Track[1]/Sample[0] by sending a SST on track_PID[1].
- 7) Host starts the transmission of Track[1]/Sample[0].
- 8) CICAM starts returning Track[0]/Sample[0].
- 9) CICAM changes the position of the Track[0]/Sample[0]/pkt[2].
- 10) Host announces the end of Track[1]/Sample[0] by sending a SET on track_PID[1].
- 11) CICAM returns the SST for Track[1]/Sample[0].
- 12) CICAM starts returning Track[1]/Sample[0].
- 13) Host announces the end of Track[1]/Sample[0] by sending a SET on track_PID[1].
- 14) CICAM sends a BLT on the comms_PID.
- 15) CICAM returns the SET for Track[0]/Sample[0].
- 16) CICAM returns the SET for Track[1]/Sample[0].

The CICAM may buffer the received TS packets. The CICAM shall inform the Host continually about the CICAM buffer fullness level as specified in clause 7.5.4.3.

After the reception of a FLT, the CICAM shall flush all the buffered TS packets and then send back the FLT unmodified on the *comms_PID*.

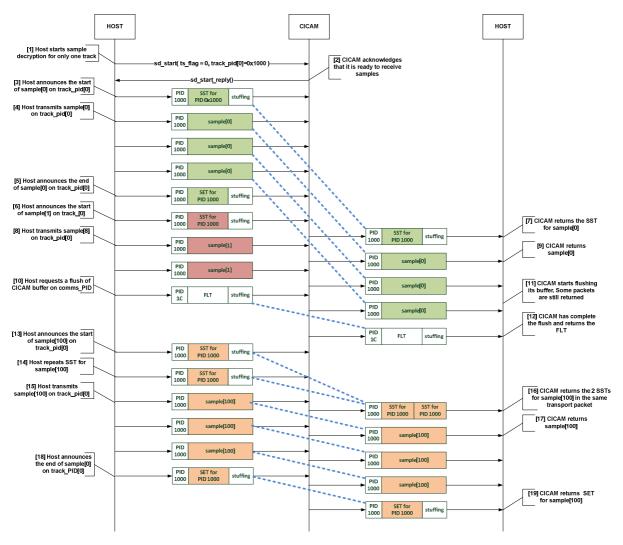


Figure 15: Example sequence with non-TS content buffer flush

NOTE 2: All PID numbers are hexadecimal.

- 1) The Host starts Sample decryption over the TS Interface by sending an $sd_start()$ APDU. There is only one Track declared (Track[0] on PID 0x1000), which is not TS ($ts_flag = 0$).
- 2) The CICAM acknowledges that it is ready to receive and decrypt Samples with an sd_start_reply() APDU.
- 3) Host announces the start of Sample[0] by sending a SST on the *track_PID*.
- 4) Host starts the transmission of Sample[0].
- 5) Host announce the end of the transmission of Sample[0] be sending a SET on the *track_PID*.
- 6) Host announces the start of the Sample[1] by sending a SST on the *track_PID*.
- 7) CICAM returns the SST for Sample[1].
- 8) Host starts the transmission of Sample[1].
- 9) CICAM starts returning the Sample[1].
- 10) Host requests a flush of the CICAM buffer by sending a FLT on the comms_PID.
- 11) CICAM starts flushing. Some pending packets are still returned to the Host until flash completion.
- 12) CICAM has completed the flush operation and sends back the FLT.

- 13) Host announces the start of Sample[100] by sending a SST on the *track_PID*, before it receives the flush confirmation from the CICAM.
- 14) Host repeats the SST.
- 15) Host starts the transmission of Sample[100].
- 16) CICAM returns the 2 SST sections in the same transport packet.
- 17) CICAM returns Sample[100].
- 18) Host announce the end of transmission of Sample[100] by sending a SET on the track_PID.
- 19) Host returns the SET on the *track_PID* after the last packet of Sample[100] has been returned.

7.5.3.4 ISOBMFF Samples

In this clause, it is detailed how the above applies to a media file of ISOBMFF content type.

7.5.3.4.1 SST Section

For the announcement of a clear Sample originating from an ISOBMFF file, the descriptor loop of the SST section shall not contain the initialization_vector_descriptor() or the key_identifier_descriptor(). Private descriptors may be present.

For the announcement of an encrypted Sample originating from an ISOBMFF file, the Host shall include exactly one initialization_vector_descriptor() and one key_identifier_descriptor() in the descriptor loop of the SST section.

The initialization_vector_descriptor() shall contain the InitializationVector of the Sample, as defined in ISO/IEC 14496-12 [8] Annex I (Common Encryption), in the *iv_data* bytes.

The key_identifier_descriptor() shall contain the KID of the Sample, as defined in ISO/IEC 14496-12 [8], Annex I (Common Encryption), in the key_id_data bytes.

7.5.3.4.2 ISOBMFF Sample packetization

Packaging of ISOBMFF Samples into a TS shall follow the rules as described above and as illustrated in figure 16.

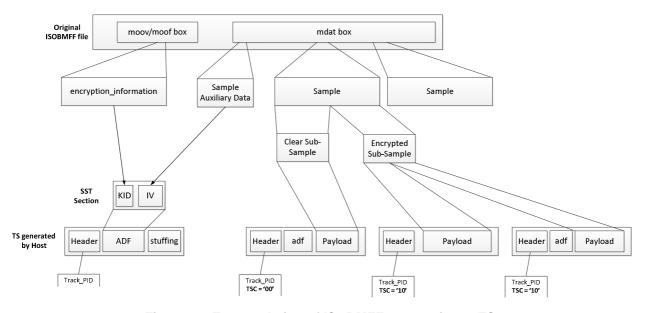


Figure 16: Encapsulation of ISOBMFF content into a TS

7.5.4 CICAM Buffering

7.5.4.1 General

When the TS Interface is configured in *Sample Mode*, the CICAM may buffer the incoming transport packets, which may then spend a varying delay in the CICAM.

7.5.4.2 Buffer size indication

The CICAM shall provide the overall size of the buffer allocated for all Sample Tracks of the TS Interface by use of the $sd_start_reply()$ APDU.

7.5.4.3 Buffer level indication

The CICAM shall provide a periodic indication of the buffer fullness level by inserting a "Buffer Level Table" (BLT) within the TS sent back to the Host. The minimum time period between consecutive BLT shall be 180 milliseconds. The maximum time period between consecutive BLT should be 220 milliseconds. The CICAM shall insert BLT sections using the *comms_PID* value of 0x001C, as designated for link-local in-band signalling in the DVB SI specification [10], clause 5.1.3. The buffer level information provided by the CICAM shall be accurate for a point in time no earlier than 100 milliseconds prior to the output of the corresponding BLT section to the Host.

It is recommended that the Host evaluates the buffer space available in the CICAM buffer for a Local TS every time a BLT section is received, in order to ensure that the CICAM buffer does not overflow when further TS packets are sent to the CICAM. The Host shall not make assumption on the buffer draining in the CICAM and shall not send more than the last known available space as indicated in the last BLT.

7.5.4.4 Flushing CICAM buffers

The Host may require the CICAM to flush all the buffered transport packets by sending a FLT on the *comms_PID* on the TS Interface. On reception of a FLT, the CICAM shall remove from its buffer all the transport packets received before the FLT. The CICAM acknowledges the flush operation completion by returning the FLT on the *comms_PID* on the TS Interface. The Host may start the transmission of some new Samples after sending the FLT without having to wait for the FLT back from the CICAM (based on the information sent by the CICAM in the last received BLT).

7.5.5 comms table

7.5.5.1 General

When the TS Interface is configured in Sample Mode, the Host and the CICAM communicate through the TS Interface itself by exchanging some *comms* tables, which are defined in this clause.

The *comms* tables specified in the present document shall be segmented into one or more private sections, applying the private section syntax defined in MPEG-2 Systems [4], before being inserted into TS packets.

TS packets containing *comms* tables that are not related to a track, i.e. FLT and BLT, shall use the PID value of 0x001C, as designated for link-local in-band signalling in the DVB SI specification [10] clause 5.1.3, and shall be mapped directly into TS packets as specified in clause 7.5.5.2.

TS packets containing *comms* tables that are related to a track, i.e. SST and SET shall use the corresponding *track_PID*, and shall be mapped into the adaptation field as specified in clause 7.5.5.3.

7.5.5.2 Mapping of comms sections into TS packets

comms sections corresponding to comms tables that are not related to a track (FLT and BLT) shall be mapped directly into TS packets. The start of the first section in the payload of a TS packet shall be pointed to by the pointer_field and the comms table may start at the beginning of the payload of the TS packet, just after the pointer_field. There is never more than one pointer_field in a TS packet, as the start of any other section can be identified by counting the length of the first and any subsequent sections, since no gaps between sections within a TS packet are allowed by the syntax.

Within TS packets of any single PID value, one section is finished before the next one is allowed to be started, or else it is not possible to identify to which section header the data belongs. If a section finishes before the end of a TS packet, but it is not convenient to open another section, a stuffing mechanism shall be used to fill up the space.

Stuffing shall be performed by filling each remaining byte of the TS packet with the value "0xFF". If the byte immediately following the last byte of a section takes the value of "0xFF", then the rest of the TS packet shall be stuffed with bytes of value "0xFF".

Stuffing shall not be performed using the *adaptation_field* mechanism.

For a more detailed description of the mechanism and functionality, refer to clause 2.4.4 and annex C of MPEG-2 Systems [4].

For a packet containing part of a *comms* section, the TS packet header bits shall be set as follows:

- The sync byte shall be set to the *LTS_id* of the Local TS.
- The transport error indicator shall be set to 0b0.
- The *payload_unit_start_indicator* of a packet containing the start of a *comms* section shall be set to 0b1, and the pointer_field shall point to the first byte of the first section starting in the TS packet. Otherwise, the *payload_unit_start_indicator* shall be set to 0b0.
- The *transport_priority* shall be 0b0.
- The *PID* shall be set to the *comms_PID* value of 0x001C, except for SST and SET, which are sent in the adaptation field of the corresponding media TS packet stream.
- The transport_scrambling_control shall be set to 0b00: comms sections shall not be encrypted.
- The *adaptation_field_control* field shall always be set to 0b01: adaptation field shall not be used for a transport packet containing part of a *comms* section.
- The *continuity_counter* shall be as defined in MPEG-2 Systems [4].

The TS packet containing the end of a *comms* section may contain the start of the next section.

There is no limitation in the maximum number of sections contained in a TS packet other than the size of the TS packet.

The CICAM shall add some BLT sections on the *comms_PID* and may modify the mapping of received *comms* sections into TS packets, but the CICAM shall always generate a compliant suite of continuity counter for all PIDs (including the *comms_PID*) at its output, except that the *continuity_counter* may be reset for a TS packet containing the start of a Sample.

7.5.5.3 Mapping of comms sections into TS packet Adaptation Fields

comms sections corresponding to *comms* tables that are related to a track, i.e. SST and SET are placed in the TS packet Adaptation Field. In this way, it is possible to transmit the beginning (SST) and end (SET) signalling in the same PID stream as the Samples themselves.

For a packet containing a *comms* section, the TS packet header bits shall be set as follows:

- The sync byte shall be set to the LTS_id of the Local TS.
- The *transport_error_indicator* shall be set to 0b0.
- The payload unit start indicator of a packet shall be set to 0b0.
- The *transport_priority* shall be 0b0.
- The *PID* shall is the same as the track_PID it is related to.
- The *transport_scrambling_control* shall be set to 0b00: *comms* sections shall not be encrypted.
- The *adaptation_field_control* field shall always be set to 0b10: this packet shall not contain any data_bytes.

• The *continuity_counter* shall be as defined in MPEG-2 Systems [4].

The contents of the Adaptation field shall be:

- Adaptation_field_length: total length of the adaptation field. As the TS packet contains only an adaptation field, the adaptation field length shall be set equal to 183 (bytes).
- Discontinuity_indicator: 0.
- Random_access_indicator: 0.
- Elementary_stream_priority_indicator: 0.
- PCR_flag: 0.
- OPCR_flag: 0.
- Splicing_point_flag:0.
- Adaptation_field_extension_flag: 0.
- Transport_private_data_flag: 1.
- private_data_byte: The first byte of the private data is the first byte of a *comms* section. The private data of a given packet shall contain an entire *comms* section. A *comms* section shall not be split over multiple TS packets. A *comms* table may be split over consecutive TS packets.

A *comms* table contained in an adaption field may contain the following descriptors from those specified in the present document and shown in table 32, and from those defined in CI PlusTM V1.3 [3]:

- ciplus_initialization_vector_descriptor.
- ciplus_key_identifier_descriptor.

The mapping of *comms* sections into TS packet Adaptation Fields is illustrated in figure 17. This figure is illustrative in that not all data elements of the TS header are depicted.

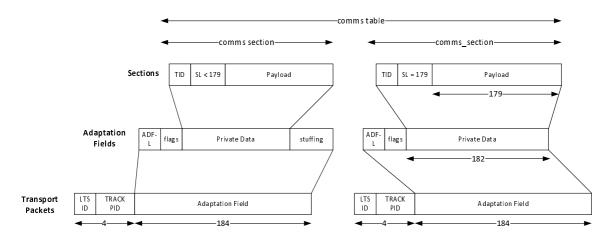


Figure 17: Mapping comms sections into TS packet Adaptation Fields

7.5.5.4 Coding of table_id values

Table 40 lists the values which shall be used for table_id for the comms table, defined in the present document.

The tag value of 0xFF is forbidden in order to avoid potential misinterpretation of stuffing bytes.

Table 40: comms table table_id values

Table Identifier	Description
0x00 to 0xCF	reserved
0xD0	sample_start_section
0xD1	sample_end_section
0xD2	flush_section
0xD3	buffer_level_section
0xD4 to 0xFE	reserved
0xFF	forbidden

7.5.5.5 Sample Start Table (SST)

The SST sections conform to the syntax for private sections as defined in MPEG-2 Systems [4].

The syntax of the SST section is shown in table 41.

Table 41: Sample_start_section syntax

Syntax	No. of bits	Mnemonic
<pre>sample_start_section(){</pre>		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserve_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
reserved	3	bslbf
tsc parity bit	1	bslbf
descriptor_loop_length	12	uimsbf
for $(i=0; i< N; i++)$ {		
descriptor()		
}		
}		

The fields are defined as follows:

table_id: This is the identifier of a sample_start_section; it shall be set to 0xD0.

section_syntax_indicator: This is a one-bit indicator which shall be set to 0b0.

section_length: This field specifies the number of bytes of the section, starting immediately following the *section_length* field and up to the end of the section. It is a 12-bit field, but the maximum length of the section shall be 179 bytes in order to fit inside a single TS packet Adaptation field, hence the first 4 bits shall be set to 0b0000.

tsc_parity_bit: This is a one-bit field which indicates the parity of the *transport_scrambling_control* to be used for the TS packet containing encrypted data of the corresponding Sample. When the *tsc_parity_bit* is set to 0b0 then the *transport_scrambling_control* shall be set to 0b10 in the header of an encrypted TS packet. Similarly when the *tsc_parity_bit* is set to 0b1 then the *transport_scrambling_control* shall be set to 0b11.

descriptor_loop_length: This 12-bit field gives the total length in bytes of the descriptors that follow.

The descriptors within the SST, if any, shall be one or more of those listed in table 45. All descriptors follow the standard descriptor format of tag value followed by a length field followed by the descriptor data.

7.5.5.6 Sample End Table (SET)

The SET sections conform to the syntax for private sections as defined in ISO/IEC 13818-1 [4].

The format of the SET section is shown in table 42.

Table 42: Sample_end_section syntax

Syntax	No. of bits	Mnemonic
<pre>sample_end_section(){</pre>		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserve_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
reserved	4	bslbf
descriptor_loop_length	12	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
descriptor()		
}		
}		

table_id: This is the identifier of a sample_end_section; it shall be set to 0xD1.

section_syntax_indicator: This is a one-bit indicator which shall be set to "0".

section_length: This field specifies the number of bytes of the section, starting immediately following the *section_length* field and up to the end of the section. It is a 12-bit field, but the maximum length of the section shall be 179 bytes in order to fit inside a single TS packet Adaptation field, hence the first 4 bits shall be set to 0b0000.

descriptor_loop_length: This 12-bit field gives the total length in bytes of the following descriptors.

The descriptors within the SET, if any, shall be one or more of those listed in table 45. All descriptors follow the standard descriptor format of tag value followed by a length field followed by the descriptor data.

7.5.5.7 Flush Table (FLT)

The FLT sections conform to the syntax for private sections as defined in MPEG-2 Systems [4].

The syntax of the FLT section is specified in table 43.

Table 43: Flush section syntax

Syntax	No. of bits	Mnemonic
flush_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserve_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
reserved	4	bslbf
descriptor_loop_length	12	uimsbf
for $(i=0; \overline{i}< N; \overline{i}++)$ {		
descriptor()		
}		
]}		

The fields are defined as follows:

table_id: This is the identifier of a *flush_section*; it shall be set to 0xD2.

section_syntax_indicator: This is a one-bit indicator which shall be set to "0".

section_length: This is a 12-bit field, the first 2 bits of which shall be "00". It specifies the number of bytes of the section, starting immediately following the *section_length* field up to the end of the section.

descriptor_loop_length: This 12-bit field gives the total length in bytes of the following descriptors.

The descriptors within the FLT, if any, shall be one or more of those listed in table 45. All descriptors follow the standard descriptor format of tag value followed by a length field followed by the descriptor data.

7.5.5.8 Buffer Level Table (BLT)

The BLT sections conform to the syntax of the private section as defined in MPEG-2 Systems [4].

The format of the BLT section is specified in table 44.

Table 44: buffer_level_table syntax

Syntax	Number of bits	Mnemonic
<pre>buffer level section(){</pre>		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserve_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
buffer_level	16	uimsbf
reserved	5	bslbf
buffer_empty_flag	1	bslbf
10%_flag	1	bslbf
20%_flag	1	bslbf
30%_flag	1	bslbf
40%_flag	1	bslbf
50%_flag	1	bslbf
60%_flag	1	bslbf
70%_flag	1	bslbf
80%_flag	1	bslbf
90%_flag	1	bslbf
buffer_full_flag	1	bslbf
}		

The fields are defined as follows:

table_id: This is the identifier of a buffer_level_section; it shall be set to 0xD3.

section_syntax_indicator: This is a one-bit indicator that shall be set to "0".

section_length: This is a 12-bit field that specifies the number of bytes of the section, starting immediately following the *section_length* field up to the end of the section.

buffer_level: The current buffer level, expressed in the number of transport packets. This is the unused space in the buffer. The buffer_level measure shall correspond to the buffer level as measured by the CICAM at most 100 ms before sending this section to the Host.

buffer_empty_flag: This bit shall be set to 1 when the buffer is empty and set to 0 otherwise.

10%_flag: This bit shall be set to 1 when the buffer_level is below 10 % of the maximum and set to 0 otherwise.

20%_flag: This bit shall be set to 1 when the *buffer_level* is below 20 % of the maximum and set to 0 otherwise.

30%_flag: This bit shall be set to 1 when the *buffer_level* is below 30 % of the maximum and set to 0 otherwise.

40%_flag: This bit shall be set to 1 when the *buffer_level* is below 40 % of the maximum and set to 0 otherwise.

50%_flag: This bit shall be set to 1 when the *buffer_level* is below 50 % of the maximum and set to 0 otherwise.

60%_flag: This bit shall be set to 1 when the *buffer_level* is below 60 % of the maximum and set to 0 otherwise.

70%_flag: This bit shall be set to 1 when the *buffer_level* is below 70 % of the maximum and set to 0 otherwise.

80%_flag: This bit shall be set to 1 when the *buffer_level* is below 80 % of the maximum and set to 0 otherwise.

90%_flag: This bit shall be set to 1 when the *buffer_level* is below 90 % of the maximum and set to 0 otherwise.

For example, if the buffer is 25 % full then the 10%_flag and 20%_flag fields shall be set to 0, and all other percentage flags shall be set to 1.

buffer_full_flag: This bit shall be set to 1 when the buffer is full and set to 0 when it is not full.

7.5.5.9 Descriptors

7.5.5.9.1 General

This clause defines the descriptors that may be used within the *comms* table. Zero or more of these descriptors may be inserted into any of the tables specified in clauses 7.5.5.4, 7.5.5.5, 7.5.5.6 and 7.5.5.7, as the construct "descriptor ()".

The following semantics apply to all of the descriptors defined in this clause.

descriptor_tag: The descriptor tag is an 8-bit field that uniquely identifies the descriptor. The *descriptor_tag* values are defined in table 45.

descriptor_length: The descriptor length is an 8-bit field specifying the total number of bytes of the data portion of the descriptor following the *descriptor_length* field.

Table 45 lists the descriptors declared within the present document, giving the descriptor tag values and the intended usage within the *comms* table. A " * " character in the intersection between a descriptor and a table type indicates that the descriptor may be included in that table type.

Tag Value SST Descriptor SET FLT BLT ciplus_initialization_vector_descriptor 0xCD 0xCF ciplus_key_identifier_descriptor 0xD0 to 0xEF Reserved 0xF0 to 0xFE Host defined Forbidden 0xFF

Table 45: Comms_table descriptors

7.5.5.9.2 CI Plus[™] initialization vector descriptor

The *ciplus_initialization_vector_descriptor()*, shown in table 46, is used by the Host to provide the initialization vector associated with the following Sample.

Table 46: ciplus_initialization_vector_descriptor syntax

Syntax	Number of bits	Mnemonic
<pre>ciplus_initialization_vector_descriptor() {</pre>		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for $(i=0; i< N; i++)$ {		
IV data byte	8	uimsbf
}		
}		

The fields are defined as follows:

descriptor_tag: This 8-bit field shall be assigned the value 0xCD.

descriptor_length: The number of *IV_data_bytes* bytes following.

IV_data_byte: This is an 8-bit field. The sequence of *IV_data_byte* fields contains the IV.

7.5.5.9.3 CI Plus[™] key identifier descriptor

The *ciplus_key_identifier_descriptor()* is used by the Host to provide the content key identifier associated with the following Sample. Its syntax is shown in table 47.

Table 47: ciplus_key_identifier_descriptor syntax

Syntax	Number of bits	Mnemonic
<pre>ciplus_key_identifier_descriptor() {</pre>		
descriptor_tag	8	uimsbf
descriptor length	8	uimsbf
for $(i=0; \overline{i} < N; i++)$ {		
key id data byte	8	uimsbf
}		
}		

descriptor_tag: This 8-bit field shall be assigned the value 0xCF.

key_id_data_byte: This is an 8-bit field. The sequence of key_id_data_byte fields contains the Key Identifier.

7.5.6 URI

The CICAM shall associate some URI to a Sample decryption program, following the rules as described in the CI PlusTM V1.3 specification [3], in clause 5.7.

The CICAM shall use the program number indicated in the sd start() APDU when sending a URI message to the Host.

The Host shall enforce the URI received from the CICAM, as described in the CI PlusTM V1.3 specification [3], in clause 5.7.

During the period when the URI for a program is not yet known by the Host, e.g. immediately after the Sample decryption has been initiated, the Host shall use the initial default URI as defined in the CI PlusTM V1.3 specification [3], in clause 5.7.

8 IP delivery CICAM player mode

8.1 General

The CI PlusTM V1.3 specification [3] provides support for the delivery of IP data across the Command Interface; however this has a limited throughput. Due to the increased data rate needed for the delivery of video data, the present document adds support for the transfer of downstream IP data from the Host to the CICAM across the TS Interface. Upstream IP data to the network continues to be sent over the Command Interface.

In some Host implementations there will be limited support for the reception and decapsulation of data received over IP. In these use cases the Host exposes resources that allow a CICAM to receive UDP/TCP payload data via the Host's IP stack across the TS Interface. It is expected that this mode will be used where the Host does not support the IP delivery mechanism, i.e. the transport mechanisms used above UDP/TCP, and/or the format of the data delivered is proprietary. The following clauses refer to this mode of operation as "Hybrid" mode.

The data returned by the CICAM shall be a SPTS that the Host player is able to consume directly.

IP delivery CICAM player mode shall be used if the Host requires the CICAM to perform a transcoding operation, or the CICAM performs a watermarking operation that implies different Sample sizes between CICAM input and output, since this is not able to be accommodated in IP delivery Host player mode/Sample Mode.

The CICAM may apply CI Plus™ Content Control scrambling for the Elementary Streams generated in CICAM Player mode.

If date and time information is not available from any broadcast tuner input, the Host and CICAM may acquire date and time information via the IP network interface. The method to acquire date and time information via the IP network interface is out of scope of the present document.

The facility of Host Service Shunning, specified in clause 10 of the CI PlusTM V1.3 specification [3] as applying to broadcast services, is not applicable to services delivered via IP delivery CICAM player mode.

8.2 Player controls

To enable the CICAM to manage the flow of data, it needs to be able to know when particular user actions have taken place. For example it may need to know if the user has paused the playback of the content, so that it can stop requesting data from the server.

The CICAM may also need to disable the use of particular player controls when they are not supported by the CICAM or the streaming format, for example.

The way in which the CICAM receives player controls is dependent on the method used to initiate playback. When it is the CICAM that initiates playback, it is assumed that the CICAM has launched a CICAM AppMMI application and that application will be used to trap user keys.

When it is the Host that is requesting the CICAM to process a service or content asset, the Host shall use the CICAM Player resource specified in clause 8.15 to inform the CICAM about user actions.

8.3 Session initialisation

8.3.1 General

The consumption of a service making use of the IP delivery CICAM player mode may be initiated by the CICAM, via the use of a CICAM application, or it may be requested by the Host via an IP channel defined on the channel line up or other out of scope mechanism.

8.3.2 Host-initiated playback

In this mode the user has requested content consumption (i.e. a service or asset) that the Host is unable to fulfil. Therefore the Host checks with each available CICAM to determine if there is one that is able to fulfil the content consumption. This is achieved using the *CICAM_player_verify_req()* APDU. This check may also be done by the Host at an earlier stage, for example during the installation of a profile with IP channels using the mechanism described in clause 15.4.

If a CICAM confirms that it can fulfil the content consumption request, the Host may request the CICAM to play the service by use of the CICAM_player_play_req() APDU. The CICAM shall then send a CICAM_player_start_req() APDU to the Host, signalling both the PMT of the SPTS that the CICAM will generate and output to the Host, and the bitrates required for data transfer across the TS Interface. The Host shall respond with a CICAM_player_start_reply() APDU, which signals the Local TS that will be used for the transfer of data across the TS Interface for this player session.

The CICAM may then establish one or more LSC Hybrid connections, as defined in clause 10, using the allocated *LTS_id* for the delivery of AV data.

In addition to the hybrid connections, the CICAM may also establish additional connections for control e.g. RTSP. These connections should be made within a new LSC session.

The CICAM then receives data, decrypts and converts the input into a Local TS, being a SPTS, and then optionally encrypts it using CI PlusTM Content Control scrambling, before sending the Local TS to the Host across the TS Interface.

During content consumption, player commands may be sent by the Host to the CICAM to control playout.

When the Host wishes to terminate the player session, it shall send a *CICAM_player_stop()* APDU to the CICAM. The CICAM shall send a *CICAM_player_end()* APDU to the Host as confirmation, when the CICAM has terminated playback. The Host shall not free any allocated resources for this player session until the *CICAM_player_end()* APDU has been received. The TS Interface shall then be put back into Normal Mode, as described in clause 7.2.

8.3.3 CICAM-initiated playback

The CICAM may at any time establish a player session with the Host in response to an event outside the scope of the present document. This is performed by the CICAM making a CICAM_player_start_req() call to the Host, signalling both the PMT of the SPTS that the CICAM will generate and output to the Host, and the bitrates required for data transfer across the TS Interface to enable this content consumption request. The Host responds with a CICAM_player_start_reply() APDU, which signals the Local TS that will be used for the transfer of data across the TS Interface for this player session.

The CICAM then typically establishes one or more LSC Hybrid connections using the allocated *LTS_id*, for the delivery of AV data.

In the case where the CICAM performs direct IP data retrieval without the use of the LSC resource, the CICAM may not establish any LSC Hybrid connections, however in all other respects the interaction between the Host and CICAM are the same.

The CICAM then requests data, decrypts and converts the input into a Local TS, being a SPTS, and may encrypt the TS packets using CI PlusTM Content Control scrambling, before sending the Local TS to the Host across the TS Interface.

During content consumption the application traps key presses, which may result in interaction with the CICAM player to control trick modes, according to a protocol whose definition is out of scope of the present document.

When the CICAM wishes to terminate the player session, it sends a CICAM_player_session_end() APDU to the Host when the CICAM has terminated playback. This APDU causes the Host to free any allocated resources for this player session, and the TS Interface is back to Normal Mode.

8.4 Communication errors

If the Host detects a communication error on the IP network interface then the Host shall relay this information to the CICAM across the Command Interface using the existing LSC *comms_reply()* APDU as defined in clause 8.7.1.5 of DVB-CI [1].

Server errors announced via protocols above TCP or UDP (e.g. HTTP over TCP) shall be returned as IP packets encapsulated in the TS across the TS Interface when in Hybrid mode.

8.5 Trick mode support

Where the content format, delivery protocol and CICAM support trick modes, the CICAM may manage the data flow and content to support the requested trick mode.

The CICAM shall continue to output a compliant TS across the TS Interface at all times during a player session. For example, in the case of fast-forward trick play, the CICAM may output a sequence of I frames, along with audio frames containing silence. In the case of a pause command the CICAM may continue to send a Local TS containing repetitions of the last video frame, along with audio frames containing silence. The exact behaviour and method to manage trick modes by the CICAM is out of scope of the present document.

8.6 Session termination

8.6.1 General

A player session may be terminated for any of the reasons specified in the following clauses. The list is not exhaustive. After a session termination the Host may need to re-initialize its A/V decoding and rendering functions, to avoid undesirable video and audio artifacts caused by the loss of the input stream.

8.6.2 Unrecoverable error

This is where the CICAM has detected an error condition which results in it being unable to continue the player session. The CICAM should signal to the user details of the error, for example via an MMI application, and shall send a CICAM_player_end() APDU to the Host.

If the player session was CICAM-initiated then the CICAM shall determine the subsequent behaviour.

If the player session was Host-initiated then the Host shall determine the subsequent behaviour.

8.6.3 Termination by the user

With a Host-initiated session, the Player session may be terminated at any time by the user, including when performing some action such as channel change. Whenever the Host wishes to terminate the player session, it shall send a *CICAM_player_stop()* APDU to the CICAM, upon which the CICAM shall terminate the session by sending the *CICAM_player_end()* APDU.

The Host should not switch to any new channel in the Player session's Local TS until it has received the *CICAM_player _end()* APDU for the session from the CICAM.

With a CICAM-initiated session, a channel change action be the user on the Host shall result in the Host sending a CICAM_player_stop() APDU to the CICAM, upon which the CICAM shall terminate the session.

The Host should not send the *ca_pmt()* APDU (signalling the channel change) to the CICAM until it has received the *CICAM_player_end()* APDU for the player session from the CICAM.

The Host shall not use this Local TS for sending A/V data for any new channel and shall not send either the *ca_pmt()* APDU referencing this Local TS (signalling the channel change) nor the *sd_start()* APDU to the CICAM until it has received the *CICAM_player_end()* APDU for the session from the CICAM.

8.6.4 End of content

If the CICAM reaches the end of the content asset in a Host-initiated player session, then the CICAM shall signal to the Host that the end of the content has been reached, using the CICAM_player_asset_end() APDU.

When the Host receives the CICAM_player_asset_end() APDU it may then instruct the CICAM to terminate the player session by sending a CICAM_player_stop() APDU to the CICAM.

End-of-content behaviour with CICAM-initiated player sessions is CICAM-specific.

8.7 CICAM player mode sequence

Figure 18 depicts the overall sequence diagram for a typical CICAM player mode content playback session for Host-initiated playback. Figure 19, similarly, depicts the CICAM-initiated content playback session. Both figures are informative.

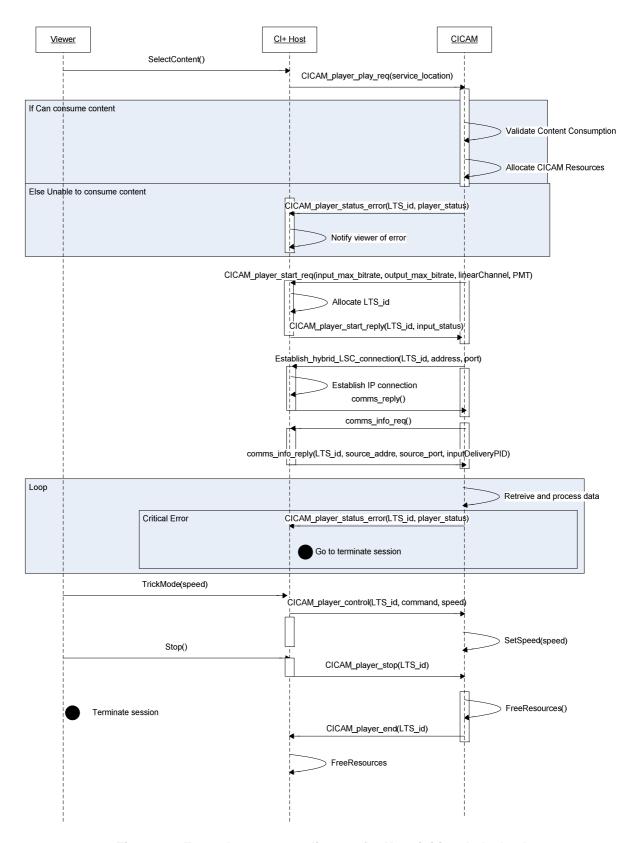


Figure 18: Example sequence diagram for Host-initiated playback

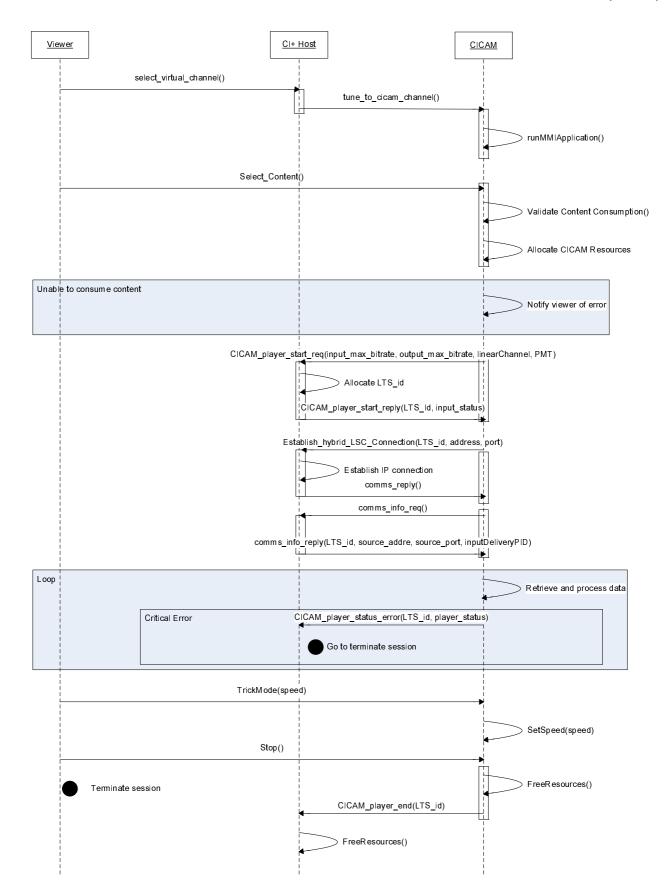


Figure 19: Example sequence diagram for CICAM-initiated playback

8.8 CICAM Player resource

8.8.1 General

The CICAM Player resource enables the Host to request the CICAM to initiate a play session and play a service on behalf of the Host. It also allows the CICAM to spontaneously request the initiation of a play session.

The CICAM Player resource is provided by the Host. The CICAM may request a session to the CICAM Player resource only if the Host announced the CICAM Player resource during the resource manager protocol (see the CI specification [1], clause 8.4.1.1). The Host may support multiple CICAM Player resource sessions.

The following clauses define the APDUs used by the Host to control content consumption by the CICAM.

NOTE: The term "player session" is used to describe a content playback session in IP delivery CICAM player mode. This is unrelated to the concept of a CI PlusTM resource session.

8.8.2 CICAM Player resource APDUs

Table 48 lists the APDUs of the CICAM Player resource and provides a brief description of the purpose of each APDU.

APDU	Direction	APDU usage
CICAM_player_verify_req	Host → CICAM	Requests CICAM to check to see if the content can
		be consumed.
CICAM_player_verify_reply	CICAM → Host	Signals if it is able to consume the content.
CICAM_player_capabilities_req	CICAM → Host	Enables the CICAM to acquire codec capabilities
		from the Host.
CICAM_player_capabilities_reply	Host → CICAM	The set of codecs that the Host supports.
CICAM_player_start_req	CICAM → Host	Request to start player session
CICAM_player_start_reply	Host → CICAM	Response to start request
CICAM_player_play_req	Host → CICAM	Request by host to play a content item.
CICAM_player_status_error	CICAM → Host	Signal critical error to Host
CICAM_player_control_req	Host → CICAM	Instructs CICAM to seek to a given position and/or
		change playback speed.
CICAM_player_info_req	Host → CICAM	Request for play speed, position, and duration
CICAM_player_info_reply	CICAM → Host	Play speed, position and duration
CICAM_player_stop	Host → CICAM	Instruct CICAM to terminate playback
CICAM_player_end	CICAM → Host	Free playback session on Host
CICAM_player_asset_end	CICAM → Host	Informs Host that end of asset has been reached
CICAM_player_update_req	CICAM → Host	CICAM requests to provide updated components
CICAM_player_update_reply	Host → CICAM	Host responds to player update request

Table 48: CICAM Player resource APDUs

8.8.3 CICAM_player_verify_req APDU

The Host may send this APDU to the CICAM to verify that the CICAM is capable of playing a content asset in IP delivery CICAM player mode. The CICAM shall reply with a CICAM_player_verify_reply() APDU.

The syntax of the *CICAM_player_verify_req()* APDU is shown in table 49.

Table 49: CICAM_player_verify_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_verify_req() {</pre>		
CICAM_player_verify_req_tag	24	uimsbf
length_field()		
service_location_length	16	uimsbf
for (i=0; i <service_location_length; i++)="" td="" {<=""><td></td><td></td></service_location_length;>		
service_location_byte	8	
}		
1 }		

CICAM_player_verify_req_tag: The tag value shall be set to 0x9FA000.

service_location_length: The number of bytes forming the ServiceLocation.

service_location_byte: These bytes form an XML data structure with a single ServiceLocation element describing the service being verified by the Host. The ServiceLocation XML schema is specified in Annex D.

8.8.4 CICAM_player_verify_reply APDU

The CICAM shall send this APDU to the Host in response to the CICAM_player_verify_req() APDU. It indicates whether the CICAM is capable of playing the IP content.

The syntax of the CICAM_player_verify_reply() APDU is shown in table 50.

Table 50: CICAM_player_verify_reply APDU Syntax

Syntax	No. of bits	Mnemonic
<pre>CICAM_player_verify_reply() { CICAM_player_verify_reply_tag length_field() = 1 player_verify_status }</pre>	24	uimsbf uimsbf

The fields are defined as follows:

CICAM_player_verify_reply_tag: The tag value shall be set to 0x9FA001.

player_verify_status: This byte returns the status of the CICAM player session. Table 51 lists the possible values.

Table 51: Possible values for play_verify_status

Program_start_status	Value
OK - service playback is possible	0x00
Error - service playback is not possible	0x01
Reserved	0x02-0xFF

8.8.5 CICAM_player capabilities_req APDU

The CICAM shall send this APDU to the Host to request the set of codecs that the Host supports. This information is used by the CICAM to determine if transcoding of the source content is required.

The syntax of the CICAM_player_play_ capabilities_req() APDU is shown in table 52.

Table 52: CICAM_player_request_capabilities APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_capabilities_req() { CICAM_player_capabilities_req_tag length_field() = 0 }</pre>	24	uimsbf

The fields are defined as follows:

CICAM player capabilities req tag: The tag value shall be set to 0x9FA002.

8.8.6 CICAM_player_capabilities_reply APDU

The Host shall send this APDU to the CICAM in response to the CICAM_player_capabilities_req() APDU, to inform about which codecs the Host supports. This information is used by the CICAM to determine if transcoding of the source content is required.

The syntax of the CICAM_player_capabilities_reply() APDU is shown in table 53.

Table 53: CICAM_player_capabilities_reply APDU syntax

Syntax	Number of bits	Mnemonic
CICAM_player capabilities_reply() { CICAM player capabilities reply tag	24	uimsbf
length_field() number of component types	16	uimsbf
<pre>for (i=0; i<number_of_component_types; content<="" i++)="" pre="" stream="" {=""></number_of_component_types;></pre>	4	uimsbf
reserved	4	bslbf
component_type	8	uimsbf
}		

The fields are defined as follows:

CICAM_player_player_capabilities_reply_tag: The tag value shall be set to 0x9FA003.

number_of_component_types: The number of component entries that follow. Where each entry represents a component that the Host is able to consume.

stream_content: This 4-bit field specifies the stream type (e.g. video, audio or EBU data) The coding of this field is the same as that used within the Component descriptor, specified in EN 300 468 [10].

component_type: This 8 bit field specifies the type of the video, audio, or EBU data. The coding of this field is the same as that used within the Component descriptor, specified in EN 300 468 [10].

8.8.7 CICAM_player_start_req APDU

The CICAM shall send this APDU in order to start a CICAM player session. The Host shall reserve bitrate across the TS Interface for the Host to CICAM, and for the CICAM to Host directions respectively, according to the information provided by the CICAM. This APDU may contain the PMT describing the components that will be delivered by the CICAM to the Host on the TS Interface, if already known. The CICAM shall in any case insert the PMT into the generated TS. The PMT shall be preceded by the PAT in the TS. If the PMT is inserted into the APDU then it should be the same as the PMT in the TS.

If the PMT is not known by the CICAM at the time the CICAM_player_start_req() APDU is sent, then the CICAM shall wait for the PMT to be determined before starting the TS delivery. When the PMT is known, the CICAM shall use the CICAM_player_update_req() APDU to inform the Host, and shall wait for the reception of the CICAM_player_update_reply() APDU from the Host. Once the PMT is known by the Host, the CICAM may start the TS transmission with the guarantee that the Host is ready to interpret the TS packets from the start of delivery of the Local TS.

The Host shall allocate a Local TS for this session and shall switch to Input Mode. If the *input_max_bitrate* is not zero (i.e. the CICAM wants to receive data from the Host on this Local TS), the CICAM shall set up a hybrid LSC connection using the *comms_cmd()* APDU (see clause 10).

The CICAM shall select appropriate PID values for the output components and for the PMT. There is no requirement for these PIDs to be the same as those allocated for the Hybrid LSC connections, if any.

The syntax of the CICAM_player_start_req() APDU is shown in table 54.

Table 54: CICAM_player_start_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_start_req() {</pre>		
CICAM_player_start_req_tag	24	uimsbf
length_field()		
input_max_bitrate	16	uimsbf
output_max_bitrate	16	uimsbf
linearChannel	1	bslbf
reserved	7	bslbf
PMT_length	16	uimsbf
for(int i=0; i <pmt_length; i++)="" td="" {<=""><td></td><td></td></pmt_length;>		
PMT_byte	8	bslbf
}		
}		

CICAM_player_start_req_tag: The tag value shall be set to 0x9FA004.

input_max_bitrate: The bitrate that the CICAM is requesting for delivery of the processed data across the Host to CICAM TS Interface, in units of 10 kbps, rounded up to the next 10kbps boundary. For example, a bit rate of 512 kbps shall be coded as the value 0x0034.

output_max_bitrate: The bitrate that the CICAM is requesting for delivery of the processed data across the CICAM to Host TS Interface, in units of 10 kbps.

linearChannel: When set signals that the session is for a linear channel, with no timeshift capabilities. When not set signals that the session is for a VOD asset or a timeshifted linear channel.

PMT_length: The number of bytes forming the PMT.

PMT_byte: A byte of the PMT, represented as an MPEG table, where the first byte is the PMT table_id.

8.8.8 CICAM_player_start_reply APDU

The Host shall send this APDU as a reply to a *CICAM_player_start_req()* APDU. The Host shall provide the Local TS identifier allocated for data delivery across the TS Interface for this session. This identifier also serves as a unique identifier for the CICAM player session.

The Host shall decide whether to create a new Local TS if one is required, before sending this APDU. If the Host is reallocating an already used Local TS, it shall stop sending TS data on this Local TS before sending this APDU. The Host may either not send any TS, including a TS clock, or send null packets until a LSC Hybrid connection is made, where the Host shall use this Local TS to send the IP data to the CICAM.

From the reception of this APDU, the CICAM shall discard any TS data on the corresponding Local TS received before reception of this APDU and may start to deliver a valid SPTS to the Host using the Local TS identifier provided by the Host.

The syntax of the CICAM_player_start_reply() APDU is shown in table 55.

Table 55: CICAM_player_start_reply APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_start_reply() { CICAM_player_start_reply_tag length_field() = 2</pre>	24	uimsbf
LTS_id input status	8 8	uimsbf uimsbf
}		

CICAM_player_start_reply_tag: The tag value shall be set to 0x9FA005.

LTS_id: The identifier of the Local TS for the player session, set by the Host. This is also used to uniquely identify the player session.

input_status: This byte returns the status of the Host. The possible values are listed in table 56.

Table 56: Possible values for input_status

input_status	Value
OK - a Local TS has switched to Input Mode	0x00
Request refused	0x01
Insufficient bitrate available	0x02
No remaining player sessions available	0x03
Reserved	0x04-0xFF

If the input_status is set to a non zero value then the *LTS_id* field shall be ignored.

8.8.9 CICAM_player_play_req APDU

The Host shall send this APDU to the CICAM to request that the CICAM plays a service on behalf of the Host. The CICAM shall reply with either a *CICAM_player_status_error()* APDU, in the case when an error is detected, or a *CICAM_player_start_req()* APDU.

The syntax of the CICAM_player_play_req() APDU is shown in table 57.

Table 57: CICAM_player_play_req APDU syntax

Syntax	No. of bits	Mnemonic
<pre>CICAM_player_play_req() {</pre>		
CICAM_player_play_req_tag	24	uimsbf
length_field()		
service_location_length	16	uimsbf
<pre>for (i=0;i<service_location_length;i++) pre="" {<=""></service_location_length;i++)></pre>		
service_location_byte	8	uimsbf
}		
}		

The fields are defined as follows:

CICAM_player_play_req_tag: The tag value shall be set to 0x9FA006.

service_location_length: The length of the ServiceLocation.

service_location_byte: These bytes form an XML data structure with a single ServiceLocation element describing the service being verified by the Host. The ServiceLocation XML schema is specified in Annex D.

8.8.10 CICAM_player_status_error APDU

The CICAM shall send this APDU to the Host in response to the CICAM_player_play_req() APDU if an error is detected. In the case where there is no error the CICAM shall send a CICAM_player_start_req() APDU instead.

It may also be sent by the CICAM at any point during player session establishment or following player session establishment if an error condition is detected.

If an error message needs to be displayed then it is a function of the CICAM to display it via an MMI application.

The CICAM shall send a CICAM_player _end() APDU to the Host to free the Host Player resources.

The syntax of the CICAM_player_status_error() APDU is shown in table 58.

Table 58: CICAM_player_status_error APDU Syntax

Syntax	No. of bits	Mnemonic
<pre>CICAM_player_status_error() {</pre>		
CICAM_player_status_error_tag	24	uimsbf
<pre>length_field() = 3</pre>		
reserved	7	uimsbf
valid_LTS_id	1	uimsbf
LTS_id	8	
player status	8	
}		

CICAM_player_status_error_tag: The tag value shall be set to 0x9FA007.

valid_LTS_id: Flag set to 1 when the error message relates to an established player session with a known *LTS_id*. If set to 0, the error message relates to the player session that is not yet established.

LTS_id: The identifier of the Local TS that is used by the player session. This is also used to uniquely identify the player session. This field is undefined if *valid_LTS_id* is set to 0.

player_status: This byte returns the status of the player session. Table 59 lists the possible values.

Table 59: Possible values for play_ status

play_status	Value
Reserved	0x00
Error - content play is not possible (e.g.	0x01
unsupported content format or protocol)	
Error - unrecoverable error	0x02
Error - content blocked (e.g. no content	0x03
license available)	
Reserved	0x04-0xFF

8.8.11 CICAM_player_control_req APDU

The Host shall send this APDU to the CICAM in order to set the position, or change the speed of a current player session. The CICAM shall respond with a CICAM_player_info_reply() APDU, which signals the requested position and speed.

The syntax of the CICAM_player_control_req() APDU is shown in table 60.

Table 60: CICAM_player_control_req APDU syntax

Syntax	Number of bits	Mnemonic
CICAM_player_control_req() {		
CICAM_player_control_req_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
Command	8	uimsbf
<pre>if(command == 0x01) {</pre>		
seek_mode	8	uimsbf
seek_position	32	tcimsbf
} else if(command == 0x02) {		
Speed	16	tcimsbf
}		
}		

The fields are defined as follows:

CICAM_player_control_req_tag: The tag value shall be set to 0x9FA008.

LTS_id: The identification of the Local TS that is used by the player session. This is also used to uniquely identify the player session.

command: identifies the action that shall take place, as specified in table 61.

Table 61: command values

Command	value
Reserved	0x00
Set position	0x01
Set speed	0x02

seek_mode: The seek mode, according to table 62.

Table 62: seek_mode

seek_mode	value
Absolute	0x00
Relative to current position	0x01

seek_position: A signed integer representing the seek position, expressed in milliseconds. A value of 0xFFFFFFF for a live stream means "jump to live", and for VoD it means the end of the content asset.

speed: The play speed expressed as hundredths of the nominal speed. A negative value means reverse speed is requested.

Examples:

- speed = 100 means nominal speed
- speed = $200 \text{ means } \times 2 \text{ in forward direction}$
- speed = $50 \text{ means } \times 0.5 \text{ in forward direction}$
- speed = 0 means pause
- speed = -100 means nominal speed in reverse direction

If the requested speed or position is not supported by the CICAM, then the CICAM shall select the nearest speed or position that it does support, or it may ignore the command. Therefore the CICAM_player_info_reply() APDU should be evaluated in order to know the actual speed, and position.

8.8.12 CICAM_player_info_req APDU

The Host shall send this APDU to the CICAM in order to get information about the currently played content, including the duration of the content, current playout position and speed. The CICAM shall reply with a CICAM_player_info_reply() APDU.

The syntax of the CICAM_player_info_req() APDU is shown in table 63.

Table 63: CICAM_player_info_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_info_req() { CICAM_player_info_req_tag length field() = 1</pre>	24	Uimsbf
LTS_id }	8	uimsbf

CICAM_player_info_req_tag: The tag value shall be set to 0x9FA009.

LTS_id: The identification of the Local TS that is used by the player session. This is also used to uniquely identify the player session.

8.8.13 CICAM_player_info_reply APDU

The CICAM shall send this APDU as a reply to a CICAM_player_info_req() APDU.

The syntax of the CICAM_player_info_reply() APDU is shown in table 64.

Table 64: CICAM_player_info_reply APDU Syntax

Syntax	No. of bits	Mnemonic
<pre>CICAM_player_info_reply() {</pre>		
CICAM_player_info_reply_tag	24	uimsbf
length_field() = 11		
LTS_id	8	uimsbf
duration	32	uimsbf
position	32	uimsbf
speed	16	uimsbf
}		

The fields are defined as follows:

CICAM_player_info_reply_tag: The tag value shall be set to 0x9FA00A.

LTS_id: The identification of the Local TS that is used by the player session. This is also used to uniquely identify the Player session.

duration: Total duration of the content in seconds. If not known, e.g. for a linear service, this shall be set to 0xFFFFFFF.

position: This signals the current play position expressed in seconds elapsed since the beginning of the content. If not known, e.g. for a linear service this shall be set to 0xFFFFFFF.

speed: This is a signed integer coding the current playout speed, expressed in hundredth of the nominal playout speed.

8.8.14 CICAM_player_stop APDU

The Host sends this APDU to the CICAM in order to terminate a player session.

The syntax of the CICAM_player_stop() APDU is shown in table 65.

Table 65: CICAM_player_stop APDU syntax

Syntax	No. of bits	Mnemonic
<pre>CICAM_player_stop() {</pre>		
CICAM_player_stop_tag	24	uimsbf
<pre>length_field() = 1</pre>		
LTS_id	8	uimsbf
}		

The fields are defined as follows:

CICAM_player_stop_tag: The tag value shall be set to 0x9FA00B.

LTS_id: The identification of the Local TS for the player session. This is also used to uniquely identify the player session.

8.8.15 CICAM_player_end APDU

The CICAM shall send this APDU in order to end a CICAM player session. This allows the Host to free up any allocated resources, e.g. bandwidth on the TS Interface. The CICAM shall close all Hybrid LSC connections used by the CICAM player session before this APDU is called.

The syntax of the CICAM_player_end() APDU is shown in table 66.

Table 66: CICAM_player_session_end APDU syntax

Syntax	Number of bits	Mnemonic
CICAM_player_end() { CICAM_player_end_tag length_field() = 1 LTS_id	24	uimsbf uimsbf
length_field() = 1	8	

The fields are defined as follows:

CICAM_player_end_tag: The tag value shall be set to 0x9FA00C.

LTS_id: The identifier of the Local TS for this player session. This is also used to uniquely identify the player session that is to be terminated.

8.8.16 CICAM_player_asset_end APDU

The CICAM shall send this APDU in order to notify the Host that either the beginning or the end of the content asset has been reached.

The syntax of the CICAM_player_asset_end() APDU is shown in table 67.

Table 67: CICAM_player_asset_end APDU syntax

Syntax	Number of bits	Mnemonic
CICAM_player_asset_end() { CICAM player session end tag	24	uimsbf
length_field() = 2	24	ullisbi
LTS_id	8	uimsbf
reserved	7	bslbf
beginning	1	bslbf
}		

The fields are defined as follows:

CICAM_player_asset_end_tag: The tag value shall be set to 0x9FA00D.

LTS_id: The identifer of the Local TS for this player session. This is also used to uniquely identify the player session.

reserved: This field shall be set to 0x7F.

beginning: When set to '1' signals that the start of the asset has been reached. Otherwise signals that the end of the asset has been reached.

8.8.17 CICAM_player_update_req APDU

The CICAM shall send this APDU in order to notify the Host of the PMT describing the components that will be delivered by the CICAM to the Host on the TS interface. This APDU shall be sent with every change of the PMT in the corresponding Local TS.

The syntax of the CICAM_player_update_req() APDU is shown in table 68.

Table 68: CICAM_player_update_req APDU syntax

Syntax	Number of bits	Mnemonic
CICAM_player_update_req() {	24	uimsbf
LTS_id PMT_length for(int i=0; i <pmt i++)="" length;="" td="" {<=""><td>8 16</td><td>uimsbf uimsbf</td></pmt>	8 16	uimsbf uimsbf
PMT_byte }	8	bslbf

CICAM_player_update_req_tag: The tag value shall be set to 0x9FA00E.

LTS_id: The identification of the Local TS that is used by the player session. This is also used to uniquely identify the player session.

PMT_length: The number of bytes forming the PMT. It shall not be zero.

PMT_byte: A byte of the PMT, represented as an MPEG table, where the first byte is the PMT *table_id*.

8.8.18 CICAM_player_update_reply APDU

The Host shall send this APDU as a reply to a CICAM_player_update_req() APDU.

From the reception of this APDU, the CICAM may deliver a valid SPTS to the Host with the guarantee that the Host is ready to interpret the transport packets from the SPTS delivery start.

Table 69: CICAM_player_update_reply APDU syntax

Syntax	Number of bits	Mnemonic
<pre>CICAM_player_update_reply() {</pre>		
CICAM_player_start_reply_tag	24	uimsbf
<pre>length_field() = 2</pre>		
LTS_id	8	uimsbf
update_status	8	uimsbf
}		

The fields are defined as follows:

CICAM_player_update_reply_tag: The tag value shall be set to 0x9FA00F.

LTS_id: The identifier of the Local TS for the player session.

update_status: This byte returns the status of the Host. The possible values are listed in table 70.

Table 70: update_status values

update_status	Value
OK - the Host has processed the updated PMT and	0x00
is ready to receive the Local TS	
Request refused	0x01
Reserved	0x02-0xFF

8.8.19 CICAM player resource summary

Table 71 provides a summary of the attributes of the CICAM player resource.

Table 71: CICAM player resource summary

Resource		Application Object		Direction			
Name	Resource identifier	Class	Туре	Ver.	APDU tag	Tag value	Host CICA M
CICAM player	00 93 00 41	147	1	1	CICAM_player_verify_req	9F A0 00	→
					CICAM_player_verify_reply	9F A0 01	+
					CICAM_player_capabilities_req	9F A0 02	←
					CICAM_player_capabilities_reply	9F A0 03	\rightarrow
					CICAM_player_start_req	9F A0 04	←
					CICAM_player_start_reply	9F A0 05	\rightarrow
					CICAM_player_play_req	9F A0 06	\rightarrow
					CICAM_player_status_error	9F A0 07	←
					CICAM_player_control_req	9F A0 08	\rightarrow
					CICAM_player_info_req	9F A0 09	\rightarrow
					CICAM_player_info_reply	9F A0 0A	←
					CICAM_player_stop	9F A0 0B	\rightarrow
					CICAM_player_end	9F A0 0C	←
					CICAM_player_asset_end	9F A0 0D	←
					CICAM_player_update_req	9F A0 0E	←
					CICAM_player_update_reply	9F A0 0F	\rightarrow

9 CICAM file retrieval

9.1 General

CICAM file retrieval is realised via a new CI PlusTM resource, namely the auxiliary file system resource. It provides a generic mechanism for a CICAM to offer files to the Host, including CICAM broadcast applications, which can be launched to run on the Host. The method used by a Host or running application to access this resource depends on the corresponding Host application environment and is out of scope of the present document. See clause 12.4.3.3 for the usage of the CICAM auxiliary file system for the provision of applications.

This new resource inherits most of its messages and underlying semantics from the Application MMI resource v2, which is defined in clause 14.5 of the CI PlusTM V1.3 specification [3].

The Host provides an auxiliary file system resource with resource identifier 0x00910041. This resource allows:

- the retrieval of files from the CICAM by the Host, for example:
 - to be used by applications running in the Host;
 - for the running application on the Host to launch an application from the CICAM;
 - for the broadcast signalling to reference an application stored on the CICAM.
- To enable the Host to gain details of available applications on the CICAM.

The file system is read-only and offers the possibility to request particular known files and receive them if they are present, otherwise an error is returned.

A CICAM offers its file system to the Host using the *FileSystemOffer()* APDU. The Host requests a file using the *FileRequest()* APDU. Files are sent by the CICAM to the Host using the *FileAcknowledge()* APDU.

The CICAM shall provide only one file system per auxiliary filesystem resource session. If the CICAM has multiple file system offerings, it shall request one session to this resource for each file system it offers.

The Host shall support a minimum of three sessions to this resource.

If the CICAM needs to stop access to its file system for any reason, for example due to a file system update, it shall close the session to the auxiliary file system resource.

An example call flow for file retrieval is depicted in figure 20.

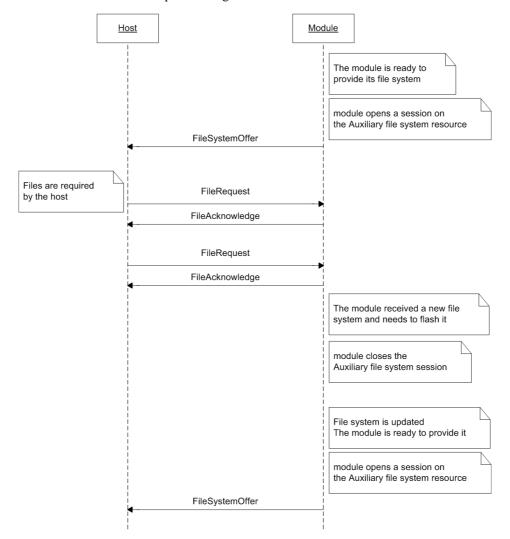


Figure 20: Example usage of the auxiliary file system resource

9.2 File system offer APDU

This APDU specifies the file system that is provided by the CICAM. Its syntax is shown in table 72.

The CICAM shall request a new session for each CICAM file system that it wishes to offer.

Only one FileSystemOffer APDU shall be sent over each session of an auxiliary file system resource opened by the CICAM.

Table 72: FileSystemOffer APDU

Syntax	Number of bits	Mnemonic
FileSystemOffer() {		
FilesystemOffer_tag	24	uimsbf
length_field()		
DomainIdentifierLength	8	uimsbf
for (i = 0; I < DomainIdentifierLength; i ++) {		
DomainIdentifier	8	uimsbf
}		
}		

FileSystemOffer_tag: This 24 bit field with value 0x9F9400 identifies this message.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

DomainIdentifierLength: This 8 bit field specifies the number of bytes of DomainIdentifier following.

DomainIdentifier: The bytes provide information needed by the Host to identify the file system provided by the CICAM. The *DomainIdentifier* may be the same as the *AppDomainIdentifier* as used by the Application MMI resource. The *DomainIdentifier* field has no specific format. The meaning of these bytes is defined by the middleware and is out of scope for the present document. Some examples of possible contents of the *DomainIdentifier* field are:

- A URL that can be used to identify the file system, and which is owned by the organisation managing the file system, for example: "www.operator.com/cifilesystem". APIs may use this value as a precursor of URLs or file paths to access the files offered by the CICAM through this resource.
- A UUID, as described in RFC 4122 [5], for example: "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6".
- A DVB-registered identifier.

9.3 File System Ack APDU

This APDU shall be sent by the Host in response to the *FileSystemOffer()* APDU to confirm whether the offered application domain is supported by the Host or not.

Table 73: FileSystemAck APDU

Syntax	No. of bits	Mnemonic
FileSystemAck() {		
FilesystemAck_tag	24	uimsbf
length_field()		
AckCode	8	bslbf
}		

FileSystemAck_tag: This 24 bit field with value 0x9F9401 identifies this message.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

AckCode: This 8 bit field communicates the response to the *FileSystemOffer()* APDU.

Table 74: AckCode values

AckCode	Meaning
0x00	Reserved for future use.
0x01	OK
	The application environment is supported by the Host.
0x02	Wrong API
	The application environment is not support by the Host
0x03 to 0xFF	Reserved for future use.

If the Host responds with AckCode equal to "wrong API", the CICAM shall close this session of the resource.

9.4 File request APDU

This APDU is copied from the Application MMI v2 resource and is fully described in clause 14.5.1 of CI PlusTM V1.3 [3].

It allows the Host to support file caching and to discover reqtypes supported by the CICAM (file, data, hash, ...).

9.5 File acknowledge APDU

This APDU is copied from the Application MMI v2 resource and is fully described in clause 14.5.2 of CI PlusTM V1.3 [3].

9.6 Auxiliary file system resource summary

Table 75 below provides a summary of the attributes of the auxiliary file system resource.

Application Object Direction Resource APDU Tag CICA Name Resource Class Type Vers. Tag value Host Identifier Μ FileSystemOffer Auxiliary file 00 91 00 41 145 9F 94 00 system FileSystemAck 9F 94 01 **FileRequest** 9F 94 02 FileAcknowledge 9F 94 03

Table 75: Auxiliary file system resource summary

9.7 Auxiliary file system and Application MMI resource coordination

The Auxiliary File System and Application MMI Resources both provide access to the CICAM file system. When both resources are open the Host shall use the resource according to the following rules:

- CICAM AppMMI applications and applications launched by CICAM AppMMI applications shall use the Application MMI resource.
- All other applications, including the case where the CICAM broadcast application is an MHEG-5 application accessing the CI_SendMessage (CIS) resident programme, as defined in ETSI MHEG Broadcast Profile [6] clause 11.10.11, shall use the Auxiliary File System resource.

10 Low Speed Communication resource version 4

10.1 General

This clause specifies a new version, version 4, of the Low Speed Communication (LSC) resource, which adds features to facilitate IP delivery CICAM player mode, to the LSC resource version 3 that is defined in clause 14.1 of the CI PlusTM V1.3 specification [3].

Version 4 of the LSC resource extends version 3 to add support for the following:

- Source-specific multicast connections (IGMPv3).
- Delivery of response data across the TS Interface.
- A new *comms_info()* APDU to transfer information about the LSC session from the Host to the CICAM.

- A new comms_IPconfig() APDU to transfer information about the IP network adapters of the Host to the CICAM.
- A new IP Source descriptor that allows the CICAM to send out a message reusing the IP source port of a previous session.

The LSC resource types are extended to add support for hybrid connections.

This clause also specifies additions to the Low Speed Communication IP extensions, defined in clause 14.2 of the CI PlusTM V1.3 specification [3].

In order to facilitate IP delivery CICAM player mode, the Host IP stack is required to comply with the following IETF RFCs:

- RFC 768 (UDP) [17]
- RFC 793 (TCP) [19]
- RFC 791(IPv4) [18]
- RFC 3376 (IGMPv3) [20]
- RFC 1112 (IP multicast) [21]

For this version of the resource, support for IPv6 is optional for the Host. IPv6 support on the Host shall be compliant with RFC 2460 (IPv6) [22] and RFC 4443 (ICMPv6) [23].

10.2 Host IP configuration information

For some delivery mechanisms the CICAM needs to make low-level IP network calls to obtain network information. For example it may need to:

- Perform a DNS lookup to establish a server's IP address from a DNS name.
- Perform a DHCP query to obtain the domain in which the Host is located.

Therefore the CICAM should have access to:

- IP address of Host.
- Address of DHCP server.
- Address of DNS servers.
- Network interface connection state.
- IP address of gateway.

A new APDU has been defined to enable the CICAM to obtain the Host network configuration information to enable these functions.

10.3 Information about IP streams

The present document adds support for exchanging information about the IP connection between the Host and the CICAM. A new *comms_info_request()* APDU is defined that enables the CICAM to obtain information about the established IP connection. In particular, for hybrid connections it defines the PID on which the IP data will be delivered across the TS Interface. It also defines information about the IP source address and port used by the Host for this connection on the IP network. The information about the IP source port is useful in case a transport protocol is used that requires negotiation of the data and control ports during session initialisation, such as RTSP. Further, it may be used when another connection is set up that wants to re-use the same IP source port (see below).

10.4 IP multicast

The present document adds support for multicast streams from the Host to the CICAM across the TS Interface. When the CICAM joins a multicast connection (through the *multicast_descriptor* as defined in clause 13.2.3), the Host shall send out an IGMP *join* packet and shall transfer the data received from the multicast source to the CICAM over the TS Interface. When the multicast connection is closed by the CICAM the Host shall send an IGMP *leave* packet to the server.

10.5 IP source ports

For some IP connections, it may be important that the outgoing data uses the same IP source port as used by the Host. For instance for linear TV services with Retransmission or Fast Channel Change mechanisms as defined by DVB-IPTV [16], or for NAT traversal mechanism messages sent to different destinations, data should originate from the same source port. For this purpose, the present document adds support for setting up an LSC session by a descriptor which reuses an IP source port currently in use for a concurrent LSC session.

10.6 Host to CICAM delivery

When working with a hybrid IP connection, on receipt of IP data the Host shall encapsulate the IP data into TS packets as defined in clause 10.12.4 and shall deliver the resulting TS packets to the CICAM using the TS Interface.

Implementers should note that server errors etc, resulting from CICAM requests will be delivered across the same TS Interface, whereas timeout errors etc. will be signalled across the Command Interface using existing LSC mechanisms defined in clause 8.7.1.5 of DVB-CI [1].

The CICAM shall support any required protocols for the Player session. For example, if the CICAM uses HTTP, then the HTTP client shall be resident on the CICAM, and the CICAM is responsible for all actions relating to the protocol, for example, HTTP Redirects.

10.7 CICAM to Host delivery

When a Hybrid connection has been established, the CICAM shall output to the Host an ISO/IEC 13818-1 [4] compliant TS, also respecting the TS buffer models. This TS may be protected using the existing CI PlusTM TS-level scrambling mechanism. The TS shall contain a PAT, and PMT for the service, along with the A/V components. The PAT and PMT may be repeated in the TS, but the CICAM shall insert new PMT if there is any corresponding change in the constitution of the TS.

The output from the CICAM shall be a TS that complies with the timing model as defined in MPEG-2 Systems [4]. It is envisaged that this TS will be consumed directly by the Host.

The CICAM shall ensure that the output bit rate, as measured over 100 ms, never exceeds that defined during the establishment of the player session.

Hybrid connections shall not have any impact, for example add substantial delay, to other Local TSs that may be running concurrently in multi-stream mode.

The CICAM shall use the Host Flow Control and the *comms_send()* APDU defined in [3] clause 14.1.4 to send IP data to the network.

10.8 IP flow control

In IP delivery CICAM player mode all flow control measures shall be performed by the CICAM. When working with a pull protocol, for example HTTP, the CICAM will request data segments based on its buffer model. In the case of a push protocol, the Host shall stream the data to the CICAM with minimal delay.

NOTE: Typically the bitrate allocated across the TS Interface should be significantly greater than the actual audio/video bitrate, in order to accommodate the bursty nature of the IP-delivery network.

10.9 comms_info

10.9.1 General

The present document introduces a new APDU to request information about an LSC session. It is only appropriate when the CICAM has requested an IP, hostname, hybrid or IP Source connection using the appropriate *connection_descriptor_type*.

The CICAM shall use this APDU when establishing a hybrid connection after it has received a *comms_reply()* APDU in order to obtain details about the PID on which the IP data will be delivered. The CICAM may also send this APDU if it wishes to know the source port on which the data is delivered.

10.9.2 comms_info_req APDU

This APDU is sent from the CICAM to the Host in order to request detailed connection information about an LSC session operating in hybrid mode. Its syntax is shown in table 76.

Table 76: comms_info_req APDU syntax

Syntax | Number of bits | M

Syntax	Number of bits	Mnemonic
comms_info_req () {		
comms_info_req_tag	24	uimsbf
length_field() = 1		
}		

comms_info_req_tag: The tag field shall be set to the value 0x9F8C07.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

10.9.3 comms_info_reply APDU

This APDU shall be sent in reply to the *comms_info_req()* APDU from the Host to the CICAM to inform the CICAM about the source IP address used by the Host for this connection, and the PID used to transfer this information from the Host to the CICAM for Hybrid connections. The syntax is shown in table 77.

Table 77: comms_info_reply APDU syntax

Syntax	Number of bits	Mnemonic
comms_info_reply () {		
comms_info_reply_tag	24	uimsbf
length_field() = 21		
LTS_id	8	uimsbf
reserved	7	bslbf
status	1	bslbf
source_IPaddress	128	uimsbf
source_port	16	uimbsf
reserved	3	bslbf
inputDeliveryPID	13	uimsbf
}		

comms_info_reply_tag: The tag field shall be set to the value 0x9F8C08.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS_id: The identifier of the Local TS.

status: This field specifies the status of the connection. A value of 0b1 signals that a connection has been established. A value of 0b0 signals that the connection is not valid. The CICAM shall ignore the fields following the status field if the status is set to 0b0.

source_IPaddress: This field is the IP source address used by the Host for this LSC session. If the Host is unable to determine the source IP address the field shall be set to all zeros. For an IPv4 address the first 12 bytes shall be set to 0x00.

source_port: This field is the source port used by the Host for this LSC session. If the Host is not aware of the source port then this shall be set to 0x0000.

inputDeliveryPID: The PID used for the delivery of the IP data across the TS Interface for Hybrid connections. These PIDs are allocated by the Host, and shall be in the range 0x0020 - 0x1FFE. Where the LSC connection is not a Hybrid connection, this value shall be set to 0x0000. In case there are multiple concurrent LSC sessions with a hybrid connection type on the same Local TS, each shall have its own LSC session and each one of them shall be allocated a unique PID.

10.10 comms IPconfig

10.10.1 General

The present document introduces a new APDU to allow the CICAM to obtain information about the IP configuration of the Host. This includes the IP addresses, default gateway and DNS server addresses that have been assigned to the Host. This information may be used, for instance, by the CICAM to directly send queries, and lookups, to the Host configured DNS server by opening an LSC session with an IP_descriptor.

10.10.2 comms_IP_config_req

This APDU shall be sent from the CICAM to the Host in order to request IP configuration information from the Host. Its syntax is shown in table 78.

Table 78: comms_IP_config_req APDU

Syntax	Number of bits	Mnemonic
comms_IP_config_req () {		
comms_IP_config_req_tag	24	uimsbf
length_field() = 0		
]		

comms IP config req tag: The tag field shall be set to the value 9F8C09.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

10.10.3 comms_IP_config_reply

This APDU shall be sent by the Host in reply to the *comms_IP_config_req()* APDU to inform the CICAM about the IP configuration of the Host. The Host shall report on the primary adapter, i.e. the adapter currently used by the host for IP communication. Its syntax is shown in table 79.

Table 79: comms_IP_config_reply APDU

Syntax	Number of bits	Mnemonic
comms_IP_config_reply () {		
comms_IP_config_reply_tag	24	uimsbf
length_field ()		
connection_ state	2	uimsbf
reserved	6	bslbf
physical_address	48	uimsbf
if (connection_state == 01) {		
IP_address	128	uimsbf
network_mask	128	uimsbf
default_gateway	128	uimsbf
DHCP_server_address	128	uimsbf
num_DNS_servers	8	uimsbf
for (int i=0; i <num_dnsservers; i++)="" td="" {<=""><td></td><td>uimsbf</td></num_dnsservers;>		uimsbf
DNS_server_address	128	uimsbf
}		
}		
}		

comms_IP_config_reply_tag: The tag field shall be set to the value 9F8C0A.

connection state: The state of the connection of this IP adapter. It shall take a value from the table 80.

Table 80: connection state values

Connection_state	Type value	Description
Disconnected	0x00	Network interface is inactive or disconnected
Connected	0x01	Network interface is active, i.e. is connected and has a valid IP address
Reserved	0x10-0x11	

physical_address: The MAC address for this IP network adapter.

IP_address: This field is the IP address allocated to this IP adapter.

network_mask: The subnet mask for this IP network adapter.

default_Gateway: This field is the IP address of the default gateway used for this IP network adapter.

DHCP_server_address: The IP address of the DHCP server used for this IP network adapter. If there is no DHCP server then this field shall be set to all zeros.

num_DNS_servers: The number of DNS servers used for this IP network adapter.

DNS_server_address: The IP address of the DNS server.

10.11 Comms Cmd modification

10.11.1 General

Two new connection types are added to the connection descriptor object to provide the parameters for:

- Source-specific multicast;
- The IP forward path over the LSC and return path over the TS Interface.

In addition a new field is introduced to allow the CICAM to signal the source port that a connection should use.

The *connection_descriptor* specified in table 14.5 of CI Plus[™] V1.3 [3] is modified to add descriptor types for the hybrid link descriptor and the source-specific multicast descriptor. The syntax of the *connection_descriptor* is shown in table 81.

Table 81: Connection descriptor APDU syntax

Syntax	Number of bits	Mnemonic
connection_descriptor () {		
connection_descriptor_tag	24	uimsbf
length_field()		
source_port_flag	1	bslbf
connection_descriptor_type	7	uimsbf
if (source_port_flag == 0b1) {		
IP_source_port	16	uimsbf
}		
if (connection_descriptor_type == SI_telephone_descriptor) {		
telephone_descriptor ()		
}		
<pre>if (connection_descriptor_type == cable_return_channel_descriptor) {</pre>		
channel_id	8	uimsbf
}		
if (connection_descripotr_type == IP_descriptor) {		
IP_descriptor ()		
}		
if (connection_descriptor_type == hostname_descriptor) {		
hostname_descriptor ()		
}		
if (connection_descriptor_type == hybrid_descriptor) {		
hybrid_descriptor ()		
}		
if (connection_descriptor_type == IP_multicast_descriptor) {		
IP_multicast_descriptor ()		
}		
}		

Table 82 lists the connection descriptor types.

Table 82: Connection descriptor type

connection_descriptor_type	Type value
SI_telephone_descriptor	0x01
cable_return_channel_descriptor	0x02
IP_descriptor	0x03
hostname_descriptor	0x04
hybrid_descriptor	0x05
multicast_descriptor	0x06
reserved for future use	0x07 - 0x7F

sourcePortFlag: When set to 0b1 signals that an IP_source_port is specified.

NOTE: The allocation of the MSB of the *connection_descriptor_type* for the *sourcePortFlag* maintains backward-compatibility with the CI PlusTM V1.3 [3] connection descriptor.

IP_source_port: The IP source port that should be used by the Host for this connection. This source port shall be one of the source ports that is already in use by another concurrent LSC session, as provided to the CICAM through the *comms_info_reply()* APDU.

In case the CICAM uses an *IP_source_port* that is not already in use for an LSC session, the Host may ignore this field and choose its own source port.

The destination address and port pair used in this IP connection shall be different from the destination address and port pair used in the LSC session that uses the same IP source port. This is in order to prevent two concurrent LSC sessions which use the same IP address, source and destination port combination.

10.11.2 Comms Cmd hybrid_descriptor

The syntax of the Comms Cmd hybrid descriptor is specified in table 83.

Table 83: Comms Cmd hybrid descriptor

Syntax	No. of bits	Mnemonic
hybrid_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
LTS_id	8	uimsbf
IP_connection_type	8	uimsbf
if (IP_connection_type == IP_descriptor) {		
IP_descriptor()		
}		
if (IP_connection_type == multicast_descriptor) {		
multicast_descriptor()		
}		
<pre>if (IP_connection_type == hostname_descriptor {</pre>		
hostname_descriptor()		
}		

descriptor tag: The descriptor tag for the *hybrid descriptor* is 0x05.

descriptor_length: The descriptor_length is an 8-bit field specifying the total number of bytes of the data portion of the *hybrid_descriptor* following the byte defining the value of this field.

LTS_id: This signals the Local TS identifier with which this connection is associated, and therefore the Local TS on which the IP data shall be delivered. This value is returned within the *CICAM_player_start_reply()* APDU, following a *CICAM_player_start_req()* APDU. If no player session has been established with the indicated *LTS_id* then the Host shall return an error.

IP_connection_type: The type of hybrid connection being requested. It shall take a value from those listed in table 84.

Table 84: IP connection type

IP_connection_type	Type value
reserved	0x01
reserved	0x02
IP_descriptor	0x03
hostname_descriptor	0x04
reserved	0x05
multicast_descriptor	0x06
reserved for future use	0x07-0xFF

IP_descriptor: This is the same as the descriptor specified in clause 14.2.1.1 of the CI Plus™ Specification V1.3 [3].

hostname_descriptor: This is the same as the descriptor specified in clause 14.2.1.2 of CI PlusTM Specification V1.3 [3].

multicast_descriptor: See clause 10.11.3.

10.11.3 Comms Cmd multicast descriptor

The syntax of the Comms Cmd multicast descriptor is specified in table 85.

Table 85: Comms Cmd multicast descriptor syntax

Syntax	Number of bits	Mnemonic
multicast_descriptor () {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
IP_protocol_version	8	uimsbf
IP_address	128	uimsbf
multicast_port	16	uimsbf
reserved	7	bslbf
include_sources	1	bslbf
num_source_addresses	8	uimsbf
for (i=0; i <num_source_addresses;)="" i++="" td="" {<=""><td></td><td></td></num_source_addresses;>		
source_address	128	uimsbf
}		
}		

descriptor tag: This field shall take the value 0x06.

descriptor_length: the *descriptor_length* is an 8-bit field specifying the total number of bytes of the data portion of the *multicast_descriptor* following the *descriptor_tag*.

IP_protocol_version: The IP protocol version, as defined by table 86.

Table 86: IP protocol version

IP_protocol_version	Type value
reserved	0x00
IPv4	0x01
IPv6	0x02
reserved for future use	0x03-0xFF

IP_address: The IP address of the multicast service. For an IPv4 address the first 12 bytes shall be set to 0x00.

multicast_port: The multicast port to be used by the host.

includeSources: When set to 0b1 this signals that the CICAM wishes to receive IP multicast flows from any of the source IP addresses listed. When set to 0b0 this signals that the CICAM wishes to receive IP multicast flows from all source IP addresses except those listed. This field is only relevant when num_source_addresses is set to a value greater than 0.

num_source_addresses: The number of multicast source addresses that follow. In the case where this is not a source specific multicast then this will be set to 0, and the Host shall accept IP multicast flows from any source address. When no source addresses are specified the Host shall assume that IGMPv2 should be used for the multicast connection.

source_address: A source address for the multicast.

10.12 Low Speed Communications resource types modification

10.12.1 General

A new value of LSC resource type is added to support hybrid connections.

The device type field for LSC resource version 4 is defined in table 87.

Table 87: Communications Device types

Description	Value
Modems	0x00-0x3F
Serial ports	0x40-0x4F
Cable return channel	0x50
reserved	0x51-0x5F
IP connection	0x60
reserved	0x61-0x6F
Hybrid connection	0x70
reserved	0x71-0xFF

The device number shall be zero for the following device types:

- IP connection.
- Hybrid connection.

If the CICAM issues a *comms_cmd()* APDU with a connection descriptor that is not supported by the Host, the Host shall respond with a *comms_reply()* APDU with *comms_reply_id = Connect_Ack* and set the field *return_value* according to table 88.

Table 88: comms reply return values

Description	Value
OK	0x00
Reserved	0x1-0x7F
Private errors	0x80-0xFD
Connection protocol not supported	0xFE
Non-specific error	0xFF

10.12.2 CICAM Flow Control

comm_rcv shall not be used for hybrid connections.

10.12.3 Disconnection behaviour

When a disconnect is initiated by the network end-point, the Host shall transmit all pending receive buffers to the CICAM and then shall send an unsolicited *comms_reply()* APDU, as specified in clause 14.1.2 of CI PlusTM V1.3 [3].

10.12.4 Data transfer across the TS Interface

10.12.4.1 TS packet syntax

When using a Hybrid connection, the data received by the Host is delivered across the TS Interface by encapsulating the IP data in TS packets. Table 89 shows the TS packet header syntax for this encapsulation.

Table 89: TS packet syntax for Hybrid data transfer

Syntax	Number of bits	Mnemonic
transport_packet () {		
sync_byte	8	bslbf
transport_error_indicator	1	bslbf
payload_unit_start_indicator	1	bslbf
transport_priority	1	bslbf
PID	13	uimsbf
transport_scrambling_control	2	bslbf
adaptation_field_control	2	bslbf
continuity_counter	4	uimsbf
if(adaptation_field_control == '01' adaptation_field_control == '11') {		
adaptation_field()		
}		
for(i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
IP_datagram_byte	8	bslbf
}		
}		

The fields shall be set as defined by MPEG-2 Systems [4] with the following exceptions:

sync_byte: This is used to identify the Local TS on the TS Interface. This shall be set to the *LTS_id* value allocated by the Host for the Player session.

transport_error_indicator: This field shall be set to 0b0.

payload_unit_start_indicator: This field shall be set to 0b0.

transport_priority: This field shall be set to 0b0.

PID: The value of this field shall be assigned by the Host such that it is unique within the Local TS. This means that each LSC session which has a hybrid connection type and which is using this Local TS will have a different PID for the IP packets associated with that session.

transport_scrambling_control: This filed shall be set to 0b00.

IP_datagram_byte: This shall contain the payload of the TCP or UDP packet.

10.12.4.2 Adaptation field usage

The adaptation field shall only be used to enable the stuffing of TS packets when the last section of IP datagram data within the TS packet does not completely fill the TS packet. This is achieved by setting the fields as follows:

adaptation_field_length: This shall be set to the number of remaining bytes in the TS packet, since the adaptation field shall be as large as to fill that TS packet.

discontinuity_indicator: This shall be set to 0b0.

random_access_indicator: This shall be set to 0b0.

elementary_stream_priority_flag: This shall be set to 0b0.

PCR flag: This shall be set to 0b0.

OPCR_flag: This shall be set to 0b0.

splicing_point_flag: This shall be set to 0b0.

transport_private_data_flag: This shall be set to 0b0.

adaptation_field_extension_flag: This shall be set to 0b0.

for (int i=0; i< N; i++) {

stuffing_byte: This shall be set to 0xFF.

}

IP Datagram1 ΙP TCP or UDP Data Header Header Data Data Data Adaptation Data Header Header Header Header TS Packet 1 TS Packet 2 TS Packet 3 TS Packet 4

Figure 21: Mapping of IP datagram to TS packets

In the case where a further IP datagram is ready for delivery, the last TS packet in figure 21 should not contain adaptation stuffing but should contain the start of the payload from the next datagram.

11 Usage Rules Information version 3

CI PlusTM V1.3 [3], in clause 5.7.1, contains the definitions of CI PlusTM URI Versions 1 and 2. The present document specifies an extension to the CI PlusTM Usage Rules Information (URI) by defining CI PlusTM URI version 3.

To ensure interoperability with legacy devices, URI version 3 applies only to Content Control type 64, version 3 and later, or Content Control type 65, see Annex A. Content Control type 64, version 3 uses the same APDUs as version 2.

The Host may declare support for URI Version 3 during URI version negotiation only on a Content Control session that uses a resource ID as defined above. A CICAM may send a URI version 3 only when the host declared support for this.

CI PlusTM URI version 3 adds the signal "trick mode control", applicable to content with emi_copy_control set to "one generation copy is permitted", to signal trick mode control to be enabled or disabled.

In the style of clause 5.7.4 of the CI $Plus^{TM}$ V1.3 specification [3], table 76 specifies the default values for CI $Plus^{TM}$ URI Version 3, including the new field "trick_mode_control_info".

Field	Default Initial Value	
protocol version	0x03	
emi_copy_control_info	0b11	
aps_copy_control_info	0b00	
ict_copy_control_info	0b0	
rct_copy_control_info	0b0	
dot_copy_control_info	0b0	
rl_copy_control_info	0b0000000	
trick_mode_control_info	0b0	
reserved bits	0b0	

Table 90: Default values for CI Plus™ URI version 3

Table 91 defines the syntax of CI PlusTM URI version 3, which extends the existing URI versions 1 and 2 message syntax contained in clause 5.7.5.2 of CI PlusTM V1.3 [3].

Table 91: URI Version 3 message syntax

Field	Number of bits	Mnemonic
uri_message() {	OI DILO	
protocol_version	8	uimsbf
aps_copy_control_info	2	uimsbf
emi_copy_control_info	2	uimsbf
ict_copy_control_info	1	uimsbf
if (emi_copy_control_info == 00) {		diriosi
rct_copy_control_info	1	uimsbf
}	-	
else {		
reserved = 0	1	uimsbf
}		
reserved for future use	1	uimsbf
if (emi_copy_control_info == 11) {		
dot_copy_control_info	1	uimsbf
rl_copy_control_info	8	uimsbf
}		
else {		
reserved = 0x00	9	uimsbf
}		
if (emi_copy_control_info == 10) {		
trick_mode_control_info	1	uimsbf
}		
else {		
reserved = 0	1	uimsbf
}		
reserved for future use	39	uimsbf
}		

The coding and semantics of the trick_mode_control_info field is specified in the style of clause 5.7.5.3 as follows:

trick_mode_control_info: This parameter describes the trick mode inhibit bit, as defined in table 92. The rules for interpretation of the trick mode control signal in the Host are outside the scope of the present document.

Table 92: Allowed values for trick_mode_control_info

Contents	Value in binary	Comment
0x0	0	Trick mode control disabled
0x1	1	Trick mode control enabled

12 CICAM applications

12.1 General

The present clause contains the specification of three aspects related to Application MMI, CICAM applications in general and their coordination with broadcast applications:

- Clause 12.2 specifies extensions to the CI PlusTM Browser (CI PlusTM Engine profile, application domain "CIEngineProfile1"), specified in CI PlusTM V1.3 [3] clause 12.2.
- Clause 12.3 specifies extensions to the CI application life cycle specified in CI PlusTM V1.3 [3] clause 12.6. These extensions are independent of the modifications contained in clause 12.4.
- Clause 12.4 specifies an Application Coordination Framework that enables the resolution of contention between broadcast applications and CICAM applications that may both want to be launched in the Host. In doing so clause 12.3 also modifies the CI application life cycle specified in CI PlusTM V1.3 [3] clause 12.6 and clause 6.5 of TS 101 699 [2].

Clause 12.5 describes aspects around Host application environments, namely what needs to be specified by an application environment in order to utilize the CI PlusTM application provision and launching facility provided in the present document, and how the CICAM can determine which environments are supported by the Host.

12.2 CI Plus™ Browser extensions

12.2.1 InteractionChannelExtension

The CI Plus™ Engine Profile is extended to include the functionality of the InteractionChannelExtension as specified in clause 15.1.2 of the MHEG-5 Broadcast Profile specification [6], with the following exception:

• In clause 15.12.2 of the MHEG-5 Broadcast Profile specification [6], Resolution of References, the hybrid file system default mapping shall map the name "//" to "CI://" instead of "DSM://".

12.2.2 ICStreamingExtension

The CI PlusTM Engine Profile is extended to include the requirements for ICStreamingExtension including 14.2.9 of the MHEG-5 Broadcast Profile specification [6] except that those relating to audio, video and subtitle formats are optional.

It is expected that system specifications referring to the present document for CI PlusTM functionality will include separately the specification of media formats for IP-delivered content.

12.2.3 ICEncryptedStreamExtension

ICEncryptedStreamExtension, specified in clause 15.16 of the MHEG-5 Broadcast Profile specification [6] on "HTTP profile for delivering encrypted streams", is optional.

It is expected that system specifications referring to the present document for CI PlusTM functionality will include separately the specification of stream encryption for IP-delivered content.

12.2.4 Video scaling

12.2.4.1 General

In CI PlusTM V1.3 [3] video scaling as a feature is not required in the CI PlusTM Engine Profile. The present document removes this exception, thus video scaling is a required feature in the CI PlusTM Engine Profile.

The following clauses specify the consequent resulting amendments to CI Plus™ V1.3 [3].

12.2.4.2 Set of features

In clause 12.2.3, table 12.2, the row Video Scaling is removed from the list of GetEngineSupport behaviour exceptions.

Table 93 lists the *GetEngineSupport* behaviour exceptions in the CI Plus™ Engine Profile compared to the MHEG-5 Broadcast Profile specification [6].

Table 93: CI Plus™ Engine Profile exceptions compared to MHEG-5 Broadcast Profile

Feature	Notes
Caching	Not required
Scene Aspect Ratio	Not required
UniversalEngineProfile	Shall adhere to MHEG-5 Broadcast Profile [6] and
	shall support the CI Plus™ profile value

12.2.4.3 GetEngineSupport

In clause 12.2.4, Table 12.3 of the MHEG-5 Broadcast Profile specification [6], the rows *VideoScaling(Chook,X,Y)[a]* and *VideoDecodeOffset(Chook,Level)* are removed from the list of *GetEngineSupport* "feature" string exceptions.

Table 94 lists the CI PlusTM Engine Profile GetEngineSupport "feature" strings.

Table 94: CI Plus™ Engine Profile GetEngineSupport "feature" strings

String		Constraint	
Standard	Short		
MultipleAudioStreams(N)	MAS(N)	May return "true" for N≤1	
MultipleVideoStreams(N)	MVS(N)	May return "true" for N≤1	
DownloadableFont(CHook)	DLF(CHook)	Shall return "true" for the values of CHook that are	
		supported by the Font class. Shall return "false" for all other	
		values of N	

12.3 Application life cycle

The CI application life cycle management tools specified in clause 12.6 of CI PlusTM V1.3 [3] are extended in the present clause to add the ability for the CICAM to enquire whether a particular application environment is supported in the Host without the Host attempting to launch the CICAM AppMMI application, and whether the launch of the CICAM AppMMI application would cause the termination of a running Host application. In order to realise this extension, the application domain identifier launch options in table 12.8 in CI PlusTM V1.3 [3] clause 12.6.2, which are applicable to the *requestStart()* APDU, are extended as shown in table 95 to include one new option, Application Domain Query (ADQ).

This extension applies to version 3 of the Application MMI resource, defined in the present document, and later versions. Apart from this extension, the set of APDUs of the Application MMI resource remains unchanged from version 2.

Table 95: Application domain identifier launch options

Name	Option Value	Notes
SSM RTGraphics State	SSM=0	Subtitles (RTGraphics) shall be disabled before the CI Plus [™] application is started, subtitles shall be returned to their existing running state when the CI Plus [™] application terminates.
	SSM=1	Subtitles (RTGraphics) shall be displayed when enabled by any user preference, if the CI Plus™ Application and subtitles are not able to coexist then the CI Plus™ Application shall not start.
	SSM=2	Subtitles (RTGraphics) shall optionally be displayed when enabled by any user preference, if the CI Plus™ Application and subtitles are not able to co-exist then subtitles shall be disabled and the CI Plus™ Application shall launch. Where the subtitle state temporarily over-rides the user preference and are disabled then the existing subtitle state shall be restored when the application terminates. This option is the default state that shall be assumed when the SSM option is omitted from the application domain specifier.
ADQ Application Domain Query	ADQ=0	The Host shall ignore the initial object in the RequestStart message. The application execution environment shall not attempt to load and execute an application. If the application domain is supported and launching an application with this application domain would not cause the termination of any other running application, then RequestStartAck shall return an AckCode of 0x01. If the application domain is not supported then RequestStartAck shall return an AckCode of 0x02. If the application domain is supported and launching an application with this application domain may cause the termination of any other running application, then RequestStartAck shall return an AckCode of 0x04. The ADQ option is available in Application MMI type 1 version 3 and later. It shall not be used in a session to Application MMI type 1 versions 1 or 2. The ADQ option is available in Application MMI type 2 (for multi-stream operation, defined in clause 6.4.6) version 1 onwards.

The AckCode values returned by the RequestStartAck() APDU, defined in TS 101 699 [2] clause 6.5.3, are extended and clarified in table 96.

Table 96: AckCode values

Meaning
Reserved for future use.
OK
If option ADQ is missing, then the application execution environment shall attempt to load and execute the initial object specified in the <i>requestStart()</i> APDU.
If option ADQ is present and ADQ=0, then the application domain is supported and launching an application with this application domain would not cause the termination of any other running application.
Wrong API
Application domain not supported.
API busy
Application domain supported but not currently available.
API clash
If option ADQ is present and ADQ=0, then the application domain is supported and launching an application with this application domain may cause the termination of some other running application.
Reserved for use with the type 2 resource for multi-stream operation, see clause 6.4.6 and table 24 in clause 6.4.6.3.
Reserved for future use.
Domain specific API busy
Application domain specific responses equivalent to response 0x03 but providing application domain specific information on why the execution environment is busy or not available for some other reason such as resource contention, when it will become available, etc.

12.4 Application Coordination Framework

12.4.1 General

CI PlusTM Host devices typically provide one or more application environments that can load and run applications that are either delivered via the broadcast channel or from elsewhere via a link communicated in broadcast signalling. These applications are referred to as broadcast applications. Contention may occur between broadcast applications and CICAM AppMMI applications (i.e. those launched via the CI PlusTM Application MMI resource) that may both request to run in the Host.

The Application Coordination Framework, defined in the present clause (12.4) and referred to hereafter as "the framework", enables the Host to make intelligent decisions about which application to launch in the event of contention between a broadcast application and a CICAM AppMMI application requesting to run in the Host. This is done in light of input from the user, broadcast signalling and negotiation with the CICAM. Thus the present document does not specify a simple fixed priority between CICAM AppMMI applications and broadcast applications.

Clause 12.4.2 contains relevant amendments, in line with the overall framework, to referenced specification clauses that could be interpreted to be prescriptive about application priority.

The framework and corresponding broadcast signalling apply only when the Host is presenting DVB content, i.e. content that is either broadcast content (whether being descrambled by the CICAM or not) or IP-delivered content destined for the CICAM. It does not apply, for example, when the Host is presenting content supplied at an HDMI[®] input, from a built-in IPTV client with embedded security, etc., or when it is running an application that is neither a broadcast application nor a CICAM AppMMI application (e.g. the native EPG, or a built-in application such as Skype). In these circumstances, a Host may implement a mechanism that permits asking the user to decide if a CICAM AppMMI application can be launched and take focus, (e.g. a mechanism conceptually like Android notifications); otherwise the CICAM AppMMI application will not be started. If there is no contention with other applications, then the Host should launch a CICAM AppMMI application that the CICAM is requesting to start.

The Host decision making process shall include input from the user and take the available applications into account. This will improve the user experience and allow them to choose, when they have indicated that they want to exercise a choice about which application to start if there is contention.

An underlying principle is that running applications, especially those being interacted with by the user, shall not be terminated without the user's consent. The present document does not specify the conditions that enable the Host to determine whether the user is in the process of interacting with a running application, since these may vary according to the application environment concerned. The Host may also include a step of verification with the user in the case that the framework or user preferences cause one application to be terminated in favour of the application that is deemed to have priority. Nothing shall prevent the user stopping one application and starting another whenever they wish.

The Host shall enable the user to indicate their preferences, including but not necessarily limited to, broadcast or CICAM AppMMI application priority. The means to do so are out of scope of the present document. At the time when a CICAM is inserted into a Host, the Host shall give the user an opportunity to indicate their preferences regarding broadcast or CICAM AppMMI application priority in a way that offers a fair and non-discriminatory choice. The selected preference shall not be unalterable once it has been set, rather it shall be enabled to be revisited easily if the user wants to change any previously taken decision. Such user preferences, if set accordingly, shall override the framework rules for resolving contention. If the user does not express a preference then the default rules defined within the framework defined in the present clause (12.4) shall apply.

The embodiment of the framework in technical terms is provided in clause 12.4.4.

Clause 12.4.5 provides informative examples of sequence diagrams to illustrate the framework.

12.4.2 Modifications to referenced specifications

The DVB CI extensions [2] and CI PlusTM V1.3 [3] specifications leave room for various interpretations about the relative priority of CICAM AppMMI applications and broadcast applications that want to run on the Host. Hence the present document makes the following changes in these referenced specifications.

In CI PlusTM 1.3 [3]:

- 1) In general clause 12 shall apply only to the MHEG-5 based application domain "CIEngineProfile1". Nothing in that clause, including clauses 12.6.2.1 and 12.6.2.2, shall apply to other application domains. Further specific changes follow in points 2) to 4) below.
- 2) In clause 12.1, in the following paragraph:

"The CI Plus Application MMI may operate in a Host that supports other application environments e.g. MHEG-5, MHP, etc. The Host implementation of the CI Plus Application MMI may elect to support the interface using any existing MHEG-5 application environment or with a separate implementation instance. The CI Plus Application MMI shall take precedence over any existing application environment and may optionally be presented on the Host native graphics plane, application plane or another display plane that may exist between the Host display and application, this is shown as a number of conceptual planes in Figure 12.3".

The sentence including the reference to "shall take precedence" shall be replaced with the following:

"When started, a CICAM AppMMI Application shall have input focus and shall either be shown in front of the broadcast applications or the broadcast applications shall not be shown. It may optionally be presented on the Host native graphics plane, application plane or another display plane that may exist between the Host display and application, this is shown as a number of conceptual planes in Figure 12.3".

3) In clause 12.3.3, the following paragraph does not apply in the context of the present document:

"The CI Plus Application shall have input focus and display priority if the CI Plus Application MMI co-exists with any other application engine (i.e. running simultaneously)."

4) In clause 12.6.2.2, the following paragraph;

"Where the broadcast profile supports MHP then the CI Plus Application MMI shall take priority over the MHP application environment and shall have input focus. The MHP graphics plane may be either temporarily removed or the CI Plus Application MMI shall appear in front of it."

shall be replaced with:

When started, a CICAM AppMMI Application launched via App MMI shall have input focus and shall either be shown in front of the broadcast applications or the broadcast applications shall not be shown.

In TS 101 699 [2]:

- 1) In clause 6.5.2, the following text is completely replaced in the present document by clause 12.4.4 below:
 - "1) kill any application currently executing on the application execution platform requested by the RequestStart;".

12.4.3 CICAM application launch mechanisms

12.4.3.1 General

Two separate mechanisms for starting applications provided by the CICAM are defined.

The first is for CICAM AppMMI applications, as specified in TS 101 699 [2], and qualified in the context of the Application Coordination Framework in the present clause 12.4.

The second, specified in clause 12.4.3.3 below, is for a new kind of application, the CICAM broadcast application.

12.4.3.2 CICAM AppMMI applications

The CICAM uses the *requestStart()* APDU to request the Host to start an application. In this case, the files of the application are delivered using the Application MMI resource as defined in TS 101 699 [2].

12.4.3.3 CICAM broadcast applications

12.4.3.3.1 General

A CICAM broadcast application is an application that is provided on the CICAM auxiliary file system (specified in clause 9) and that the Host launches after reading the corresponding file. A CICAM broadcast application may be launched following a request by broadcast signalling or a running broadcast application.

12.4.3.3.2 Signalling CICAM broadcast applications in the broadcast stream

DVB defines a method for broadcast application signalling in TS 102 809 [7] and TS 102 727 [i.1]. This is extended to include broadcast signalling for CICAM broadcast applications, as specified in annex F. CICAM broadcast applications that are signalled within a system using the aforementioned DVB specifications shall be signalled as defined in clause F.2.

This extended DVB application signalling provides a mechanism whereby the respective priorities of broadcast and CICAM broadcast applications can be signalled. It enables the use case, for example, of providing a default version of an application in the broadcast channel, but allowing a richer version of the application stored on the CICAM auxiliary file system to take priority, for users that have the corresponding CICAM installed in the Host.

It is expected that the contents of annex F, or a further development thereof, will be incorporated into a future revision of the DVB application signalling specification [7]. When that revision has been published, annex F of the present document shall be considered as being deprecated. A future revision of the present document may have annex F removed and refer directly to the revised DVB application signalling specification [7] for the specification of the method to advertise the availability of CICAM broadcast applications.

Where both broadcast and CICAM broadcast applications are signalled in the broadcast application signalling for a service, the priority setting mechanism, if defined explicitly in that system, or alternatively some other mechanism defined within the specification of the broadcast application signalling, shall be used to decide which shall be started.

Other solutions for broadcast application signalling may also be applicable but these are outside the scope of the present document.

12.4.3.3.3 Advertising CICAM broadcast applications

The present clause specifies the method by which the CICAM advertises applications stored on its auxiliary file system (see clause 9) to the Host.

Each CICAM broadcast application provided shall be exposed to the Host through an XML AIT file (see clause 5.4 of TS 102 809 [7]), which is also stored on the auxiliary file system, with contents populated as follows:

- The XML file shall contain an application discovery record containing one or more application elements, all with the same *orgId* and *appId* values and all containing the application type corresponding with the application domain of the auxiliary file system in which the XML file resides.
- The *name* element shall be encoded in at least one language.
- The location of the initial object of the CICAM broadcast application shall be indicated through the
 applicationLocation element, which shall be set as defined in the convention specified below in the present
 clause.
- The *applicationLocation* element shall reference an initial object provided by the CICAM. When the initial object is provided by the auxiliary file system, the domain identifier shall not be included, but shall be derived from the application type field.
- The *type* element of the application descriptor shall be encoded identifying the application environment, as defined by that environment or middleware technology used by the application (see clause 12.5.2). The application domain "CIEngineProfile1" defined in clause 12.2 of CI PlusTM 1.3 [3] and modified in the present document shall use the application type "application/vnd.dvb.ciplus.mheg".
- The icon element may be encoded.
- If the application environment defines the use of the XML AIT then the use of the other fields and elements shall be as defined by the specification for the application environment. Otherwise use of the other fields and elements is not defined.

NOTE: Use of the XML-based syntax of the AIT in these circumstances does not have any dependency or imply use of the MPEG-2 table and section syntax of the AIT in the broadcast stream.

These XML files shall be included under "/dvbapplication/" in the auxiliary file system.

The name of the files shall follow the convention of "CICAMApplication." followed by an integer. The integers shall start at one and increment by 1 in sequence without any gaps or leading zeros. For example, if a CICAM wishes to advertise three applications to the Host then it would provide files named "/dvbapplication/CICAMApplication.1", "/dvbapplication/CICAMApplication.2" and "/dvbapplication/CICAMApplication.3".

Delivery of files from the CICAM to the Host shall be supported with the following XML schema:

12.4.4 Broadcast and CICAM application coordination

12.4.4.1 General

The present clause (12.4.4) provides the technical embodiment of the Application Coordination Framework.

As described in clause 12.4.1, user preferences administered on the Host device, if set accordingly, shall override any of the assertions relating to application priority in the present clause (12.4.4).

Clause 12.4.4.2 asserts the framework application priority rules around the situation when the Host changes to a DVB service.

Clause 12.4.4.3 asserts the framework application priority rules around the situation when either a broadcast application or a CICAM AppMMI application is running, as a result of priority having been determined previously in the case that contention occurred, or in the case that no contention occurred.

Clause 12.4.4.4 asserts the framework application priority rules around the situation when the CICAM Virtual Channel (see clause 14) has been selected.

Clause 12.4.4.5 asserts the framework mechanisms around the termination of applications.

In general, once launched the application with priority shall have input focus, meaning it is displayed to the user and subsequent user interaction occurs within that application. The details of this are outside the scope of the present document, as they depend on the application environment or environments involved and on the Host implementation, in the case of coordinating applications in different application environments.

When an application starts and has priority over already running applications or gains priority over other running applications that it did not previously have, what happens depends on the ability of the application environments and/or the Host to run more than one application at the same time.

Hosts that have the capability to run multiple applications and/or application environments at the same time may include a mechanism to allow the user to switch focus between applications. Use of this mechanism will result in the application that formerly had priority losing input focus and may result in other applications being shown in front of it, or it not being shown.

Any applications that the Host cannot run at the same time as the one with priority shall be terminated, subject to any optional verification with the user. The reasons why a Host cannot run more than one application at the same time are outside the scope of the present document, but some possible examples include:

- the application with priority uses an application environment that supports running only one application at a time; or
- the Host is unable to run other application environments at the same time as the application environment needed for the application that has priority.

Stopping one application environment and starting another may take significant time and result in a poor user experience. In markets where broadcast applications are deployed, it is recommended that CICAMs use the same application environment for CICAM AppMMI applications as are used for broadcast applications in that market.

12.4.4.2 Changing to a DVB service

After the change to a DVB Service has been completed and the corresponding TS is being received, the Host shall monitor the TS for broadcast application signalling, interpret the signalling provided for that DVB Service, and launch the broadcast application if and as appropriate.

The Host is said to have determined that no broadcast application will be launched if any of the following conditions apply during the chain of events leading to the launch of a broadcast application:

- No broadcast application signalling is present (in the DVB case this is a link to the AIT referenced from the PMT, and the AIT itself).
- Broadcast application signalling is present but does not result in an application being launched. This could be
 due to the Host not being able to launch the application, or a persistent error in the signalling or carriage of the
 application.

If the Host determines that no broadcast application will be launched and there are no conflicts with any other running application then CICAM AppMMI applications may be launched by the Host if requested by the CICAM.

If the CICAM requests to start an AppMMI application before the Host has completed the verification that it will not be launching a broadcast application, which may take several seconds, then the Host shall not respond to the *RequestStart()* APDU from the CICAM until this verification is complete. When the verification is complete the Host shall respond with the *RequestStartAck()* APDU as specified in clause 12.3, according to whether the AppMMI application will be launched or not.

If an auto-start broadcast application is launched, then requests from the CICAM to start a CICAM AppMMI application shall be denied by the Host. This shall be reported back to the CICAM with a *requestStartAck()* APDU with an AckCode value of 0x03 ("API busy", see clause 12.3) or with a "Domain specific API busy" value if defined for the application domain. See clause 6.5.3 of TS 101 699 [2].

Application persistence across DVB Service changes is as defined by the specification for that application environment. The present document specifies the facility to assert the persistence of a CICAM AppMMI application across a service change, as specified in clause 13.2.1.

Where both conventional broadcast applications and CICAM broadcast applications are signalled in the broadcast signalling for a service, the priority indication, or similar, as specified for the signalling system shall be used to decide which shall be started. Annex F includes this mechanism as an extension of DVB broadcast application signalling [7].

12.4.4.3 An application is running and has focus

If a CICAM AppMMI application is running then the Host shall not start a broadcast application except as part of the process for changing to a DVB Service as defined above.

This implies that a broadcast application middleware would be prevented from obeying changes in application signalling, for example, if a new autostart application became signalled while a CICAM AppMMI application is running. The Host may, however, optionally include the facility to notify the user of the broadcast application having become available, by means that are outside the scope of the present document.

A running broadcast application may launch a CICAM broadcast application at any time, under the control of the corresponding application environment middleware running in the Host. Likewise, a running CICAM broadcast application may launch another broadcast application or CICAM broadcast application at any time.

There is no mechanism defined for a broadcast application to launch a CICAM AppMMI application.

12.4.4.4 Changing to the CICAM Virtual Channel

Clause 14.2.1 of the present document defines the use of Application MMI when entering the Virtual Channel. When the user chooses to start the Virtual Channel, the application identified in the *requestStart()* APDU shall have user focus and any running broadcast or CICAM application may lose focus or be terminated, depending on the capabilities of the Host.

12.4.4.5 enter_menu

The Host sends the *enter menu()* APDU to the CICAM as a result of the user choosing to enter the CICAM menu.

If the CICAM responds to <code>enter_menu()</code> with a <code>requestStart()</code> APDU then the application identified in the <code>requestStart()</code> APDU coming from the CICAM shall have user focus and any running broadcast or CICAM application may lose focus or be terminated, depending on the capabilities of the Host.

NOTE: Depending on the CICAM implementation, the CICAM could launch High Level MMI after receiving the *enter menu()* APDU from the Host.

12.4.4.6 Application termination

An application environment need not comply with requirements in CI PlusTM 1.3 [3] or TS 101 699 [2] for the Host to terminate a CICAM AppMMI application following a service change if all of the following apply;

- the application environment supports the identification of applications; and
- the application environment supports application persistence across service changes; and
- the CICAM AppMMI application is permitted to run in the new service according to the broadcast application signalling in that service.

12.4.5 Application coordination scenarios

12.4.5.1 General

Figures 22, 23 and 24 provide informative examples, in the form of annotated sequence diagrams, of three typical scenarios of operation of the Application Coordination Framework.

12.4.5.2 Broadcast signalled application launch

Figure 22 depicts an example scenario of the launch of broadcast signalled applications.

The operator provisions the CICAM with one or more applications. Applications are associated with an application domain. The mechanism by which the operator provisions these applications to the CICAM is out of the scope of the present document.

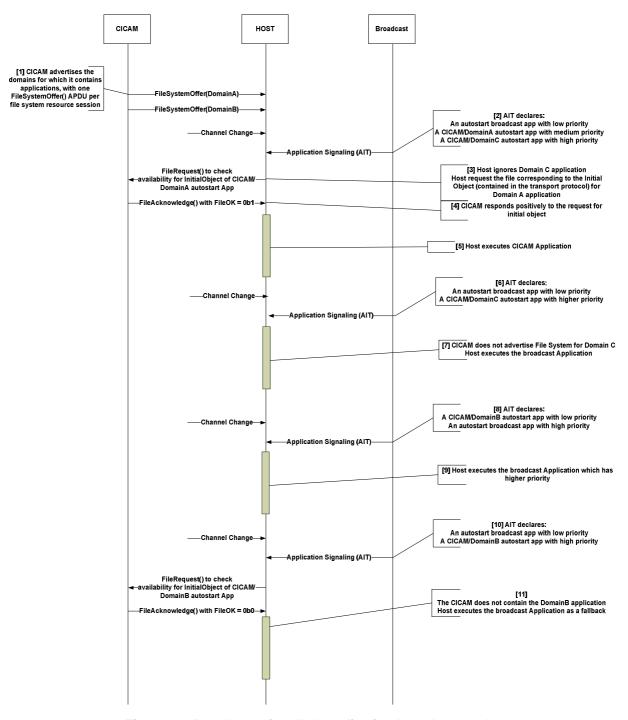


Figure 22: Broadcast signalled application launch example

- 1) The CICAM advertises the application domains "DomainA" and DomainB", for which it is providing applications, by use of the Auxiliary File System resource, sending one FileSystemOffer() APDU per file system resource session.
- 2) The Host changes to a channel for which the AIT declares:
 - a) An autostart broadcast application with low priority.
 - b) An autostart "DomainA" application located on CICAM with medium priority.
 - c) An autostart "DomainC" application located on CICAM with medium priority.
- 3) Host ignores the signalling of "DomainC" application as "DomainC" is not advertised by the CICAM. Host requests from the CICAM the Initial Object of the "Domain A" application by use of the *FileRequest()* APDU.

- 4) The CICAM returns positively the requested Initial Object with the *FileAcknowledge()* APDU.
- 5) Host executes the "DomainA" application.
- 6) The Host changes to a channel for which AIT declares:
 - a) An autostart broadcast application with low priority.
 - b) An autostart "DomainC" application located on the CICAM with medium priority.
- 7) Host ignores the signalling of "DomainC" application and launches the broadcast application.
- 8) The Host changes to a channel for which AIT declares:
 - a) An autostart "DomainB" application located on the CICAM with low priority.
 - b) An autostart broadcast application with high priority.
- 9) Host launches the broadcast application, which has priority.
- 10) The Host changes to a channel for which AIT declares:
 - a) An autostart broadcast application with low priority.
 - b) An autostart "DomainB" application located on CICAM with high priority.
- 11) Host request to the CICAM the Initial Object of the "Domain B" application by use of the *FileRequest()* APDU. The CICAM file system does not contain the request file. The CICAM responds negatively to the file request. The Host understands that the CICAM file system does not contain the application and decides to launch the broadcast application instead.

12.4.5.3 CICAM AppMMI application launch

Figure 23 depicts an example scenario of the launch of CICAM AppMMI applications.

The operator provisions the CICAM with one or more applications. Applications are associated with an application domain and an XML AIT file. The mechanism by which the operator provisions these applications to the CICAM is out of the scope of the present document.

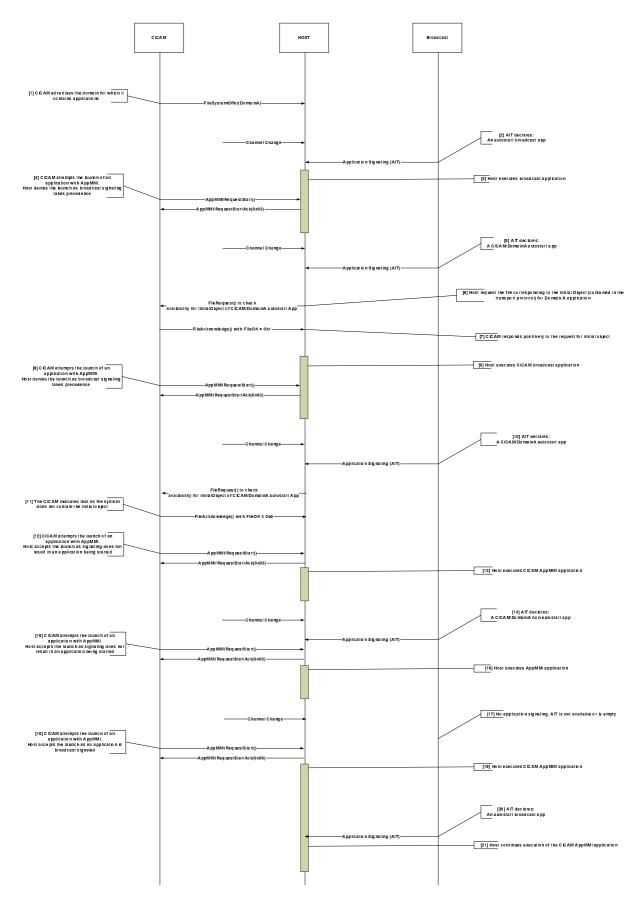


Figure 23: CICAM AppMMI application launch example

1) The CICAM advertises the application domain "DomainA" for which it embeds applications by use of the Auxiliary File System resource (*FileSystemOffer()* APDU).

- 2) The Host changes to a channel for which AIT declares an autostart broadcast application.
- 3) Host starts the broadcast application.
- 4) The CICAM attempts to launch an application by use of the *RequestStart()* APDU. Host denies the launch as the broadcast signalling takes precedence.
- 5) The Host changes to a channel for which AIT declares an autostart "DomainA" application located on the CICAM file system.
- 6) Host request the initial object corresponding to the signalled application.
- 7) CICAM returns positively the requested initial object.
- 8) Host launches the CICAM broadcast "DomainA" application.
- 9) The CICAM attempts to launch an application by use of the *RequestStart()* APDU. Host denies the launch as the broadcast signalling takes precedence.
- 10) The Host changes to a channel for which AIT declares an autostart "DomainA" application located on the CICAM file system.
- 11) The CICAM indicates that its file system does not contain the initial object for the "DomainA" application.
- 12) The CICAM attempts to launch an application by use of the *RequestStart()* APDU. Host accepts the launch as the broadcast signalling does not result in an application being started.
- 13) Host executes the CICAM AppMMI application.
- 14) The Host changes to a channel for which AIT declares a non-autostart "DomainA" application located on the CICAM file system.
- 15) The CICAM attempts to launch an application by use of the *RequestStart()* APDU. Host accepts the launch as the broadcast signalling does not result in an application being started.
- 16) Host executes the AppMMI initiated application.
- 17) The Host changes to a channel with no application signalling.
- 18) The CICAM attempts to launch an application by use of the *RequestStart()* APDU. Host accepts the launch as no application is broadcast signalled.
- 19) Host executes CICAM AppMMI application.
- 20) An update of the application signalling indicates availability of an autostart broadcast application.
- 21) The Host continues the execution of the AppMMI initiated application.

12.4.5.4 Broadcast application launches CICAM broadcast application

Figure 24 depicts an example scenario of the launch of CICAM AppMMI applications.

The operator provisions the CICAM with one or more applications. Applications are associated with an application domain and an XML AIT file. The mechanism by which the operator provisions those applications to the CICAM is out of the scope of the present document.

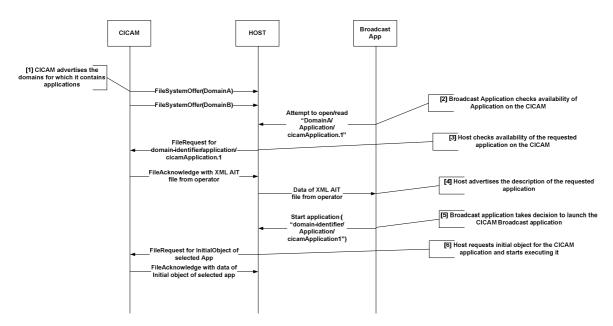


Figure 24: Broadcast application launches CICAM broadcast application example

- 1) The CICAM advertises the application domain "DomainA" for which it provides applications by use of the Auxiliary File System resource (*FileSystemOffer()* APDU).
- The Host is executing a broadcast application which checks for availability of an application on the CICAM. The interface for the broadcast application to request the Host to perform the check is out of the scope of the present document.
- The Host checks availability of the requested application XML AIT on the CICAM by use of the FileRequest() APDU.
- 4) The Host confirms availability of the CICAM application to the executing application and advertises the attributes of that application.
- 5) The executing application requests the Host to start the execution of the CICAM Application. The interface for the broadcast application to request the Host to execute the CICAM application is out of the scope of the present document.
- 6) The Host requests the initial object of the CICAM Application and starts it.

12.5 Host application environments

12.5.1 General

Clause 12.5.2 notes what needs to be defined by an application environment specification that intends to make use of the framework to provision and launch applications from the CICAM.

Clause 12.5.3 lists the methods by which the CICAM is able to determine the application environments supported by the Host.

12.5.2 Application provision on a CICAM

In order for a middleware system or application environment to include the facility to provision and launch applications that are compliant with that system from a CICAM according to the present document, the middleware system needs to define an identification of the middleware system and an object which allows the Host to launch the application.

The middleware is identified via the "AppDomainIdentfiier" (TS 101 699 [2] clause 6.5.2) and the "DomainIdentifier" (clause 9.2). A middleware shall define values for these two identifiers. For the MHEG-based CI PlusTM browser, the value of the "AppDomainIdentfier" and "DomainIdentfier" is "CIEngineProfile1".

The object which allows a Host to launch an application is referred to as the "InitialObject" (TS 101 699 [2], clause 6.5.2). A middleware shall define an appropriate value for this object. For the MHEG-based CI Plus™ browser, the "InitialObject" is a URI. If the middleware works with URIs, then it should create a URI that points to the CICAM. In the case of MHEG, the URI that points to the CICAM is: CI://

12.5.3 Determining Host application environment support

A CICAM is able to discover which application environment(s) a Host supports in two ways:

- By the *FileSystemAck()* APDU *AckCode* value received from the Host for each file system offer that the CICAM issued to the Host. This is specified in clause 9.
- By using the *RequestStart()* APDU with the "ADQ=0" parameter and the known Application Domain Identifier of the application environment whose support is being queried, as specified in clause 12.3.

13 DVB Host Control version 3

13.1 General

DVB Host Control resource version 3 (with resource ID 0x00200043) is an extension to version 2 and adds the following features:

- The capability to tune directly to a logical channel number, including the CICAM Virtual Channel and IPdelivered services.
- The capability to tune directly to an IP location.
- The capability to tune directly to a DVB triplet.
- The capability to keep an application launched by Application MMI running after a channel change initiated by the CICAM.
- The query of the available delivery systems supported by the Host.

The CICAM shall not keep this resource open when it has not issued one of the tune request APDUs.

13.2 DVB Host Control version 3 APDUs

13.2.1 tune_broadcast_req APDU

The *tune_broadcast_req()* APDU defined in clause 14.6.2.1 in CI PlusTM V1.3 [3] is extended as shown in table 97. The Host shall respond to this with a *tune_reply()* APDU.

Table 97: tune broadcast reg APDU

Syntax	Number of bits	Mnemonic
tune_broadcast_req() { tune_broadcast_req_tag length_field() reserved tune_quietly_flag keep_app_running_flag pmt_flag service_id reserved descriptor_loop_length for (i=0; i <n; (pmt_flag="=" 1)="" descriptor()="" i++){="" if="" program_map_section()<="" td="" {="" }=""><td>5 1 1 1 1 16 4 12</td><td>uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf</td></n;>	5 1 1 1 1 16 4 12	uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf uimsbf
}		

The fields are defined as follows:

tune_quietly_flag: The *tune_quietly_flag* is a 1 bit flag signalling whether the service change shall be indicated to the user. When the flag is set, implementation dependant behaviour such as the now/next banner and front panel display shall be suppressed and any subsequent navigation with channel up/down keys shall be relative to the originating service

keep_app_running_flag: The *keep_app_running_flag* is a 1 bit flag signalling whether the requested tune should keep any application launched without any application identification running across the service tune. A value of 0b1 indicates the tune shall be performed non-destructively and attempt to keep the application running during and after the tune subject to the following restrictions:

- If there is a broadcast-signalled application that the Host can launch on the service to which the CICAM tunes, then any running application without any application identification shall be either terminated or lose focus, depending on the capabilities of the Host (see clause 12.4.4.1).
- If a running application is terminated due to the previous condition being fulfilled then the tune_quietly_flag and keep app running flag shall be ignored.
- Running applications that have application identification shall adhere to the lifecycle rules valid for the respective application environment. See clause 12.4.4 for more information.

If a running CICAM AppMMI application is terminated due to a broadcast application being signalled then the Host shall send an *AppAbortRequest()* APDU to the CICAM.

Other fields: Refer to clause 14.6.2.1 of CI PlusTM V1.3 [3] for definitions of the other parameters in the *tune_broadcast_reg()* APDU.

13.2.2 tune_triplet_req APDU

The *tune_triplet_req()* APDU replicates the *tune()* APDU defined in clause 8.5.1.1 of the DVB-CI specification [1], but it omits the network_id parameter and adds the dsd_type parameter, to enable the CICAM to specify from which broadcast system the service is to be obtained. It also adds the *tune_quietly_flag* and the *keep_app_running_flag* defined in clause 12.2.1. The Host shall respond to this with a *tune_reply()* APDU.

The syntax of the *tune_triplet_req()* APDU is shown in table 98.

Table 98: tune_triplet_req APDU syntax

Syntax	Number of bits	Mnemonic
<pre>tune_triplet_req () {</pre>		
<pre>tune_lcn_triplet_tag length field()</pre>	24	uimsbf
Reserved	6	uimsbf
tune_quietly_flag	1	uimsbf
Keep_app_running_flag	1	uimsbf
original_network_id	16	uimsbf
transport_stream_id	16	uimsbf
service_id	16	uimsbf
<pre>delivery_system_descriptor_tag if (delivery_system_descriptor_tag == 0x7f) {</pre>	8	uimsbf
<pre>descriptor_tag_extension }</pre>	8	uimsbf
else {		
reserved }	8	uimsbf

tune_triplet_req_tag: This tag shall be set to the value of 0x9F8409.

tune_quietly_flag: Refer to the syntax of the tune_broadcast_req() APDU in clause 12.2.1.

keep_app_running_flag: Refer to the syntax of the *tune_broadcast_req()* APDU in clause 12.2.1.

original_network_id: The original_network_id of the requested service.

transport_stream_id: The transport_stream_id of the requested service.

service_id: The service_id of the requested service.

delivery_system_descriptor_tag: This field specifies the delivery system type from which the service is requested. Tag values are defined in DVB SI specification [10].

descriptor_tag_extension: This field, along with the *delivery_system_descriptor_tag*, specifies the delivery system type from which the service is requested..Tag values are defined in the DVB SI specification [10].

13.2.3 tune Icn reg APDU

The *tune_lcn_req()* APDU is sent by the CICAM to the Host in order to tune the Host directly to a service by the LCN. Its syntax is shown in table 99. The Host shall respond to this APDU with a *tune_reply()* APDU.

This APDU cannot be used to tune between channel lists. The Host shall resolve the LCN according to its active channel list. If the LCN refers to a service which is unknown according to the currently active channel list, then the Host shall not honour the request and shall return a *tune_reply()* APDU with a *status_field* of 0x05 (service not found).

Table 99: tune_lcn_req APDU syntax

Syntax	Number of bits	Mnemonic
tune_lcn_req () {		
tune_lcn_req_tag	24	uimsbf
length_field()		
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
logical_channel_number	14	uimsbf
}		

tune_lcn_req_tag: This tag shall be set to the value of 0x9F8407.

tune_quietly_flag: Refer to the syntax of the tune_broadcast_req() APDU in clause 12.2.1.

keep_app_running_flag: Refer to the syntax of the *tune_broadcast_req*() APDU in clause 12.2.1.

logical_channel_number: This 14-bit field indicates the logical channel number assigned to the service. Four decimal digits can be accommodated, hence this number shall be within the range 0 to 9999 inclusive.

13.2.4 tune_ip_req APDU

The *tune_ip_req()* APDU is sent by the CICAM to the Host to provide the location of a single IP-delivered service to which the Host is requested to connect. Its syntax is shown in table 100. The Host shall respond to this with a *tune_reply()* APDU.

Table 100: tune_ip_req APDU

Syntax	Number of bits	Mnemonic
<pre>tune_ip_req () {</pre>		
tune_ip_req_tag	24	uimsbf
length_field()		
reserved	2	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
service_location_length	12	uimsbf
for $(i=0; i< N; i++)$ {		
service_location_data	8	uimsbf
]		
}		

tune_ip_req_tag: This tag shall be set to the value of 0x9F8408.

tune_quietly_flag: Refer to the syntax of the tune_broadcast_req() APDU in clause 12.2.1.

keep_app_running_flag: Refer to the syntax of the *tune_broadcast_reg()* APDU in clause 12.2.1.

service_location_length: The length of the following *ServiceLocation*.

service_location_data: A text string which describes a valid XML description containing a single ServiceLocation element conforming to the XML schema defined in annex D.

13.2.5 tuner_status_req APDU

The *tuner_status_req()* APDU is sent by the CICAM to the Host to gain knowledge of what broadcast delivery systems are supported by the Host, and the likely success of a tune depending on the delivery system type. Its syntax is shown in table 101. The Host shall reply to this APDU with the *tuner_status_reply()* APDU.

Table 101: tuner_status_req APDU

Syntax	No. of bits	Mnemonic
<pre>tuner_status_req() {</pre>		
tuner_status_req_tag	24	uimsbf
length_field() = 0		
}		

tuner_status_req_tag: This tag shall be set to the value of 0x9F840A.

13.2.6 tuner_status_reply APDU

The *tuner_status_reply()* APDU is sent by the Host in response to the *tuner_status_req()* APDU. The Host uses this APDU to inform the CICAM which broadcast delivery system or systems it supports, and which of them is likely to give a successful tune, and whether it supports IP-delivered services. The APDU syntax is shown in table 102.

Table 102: tuner_status_reply APDU syntax

Syntax	No. of bits	Mnemonic
<pre>tuner status reply() {</pre>		
tuner status reply tag	24	uimsbf
length field()		
IP tune capable flag	1	uimsbf
num dsd	7	uimsbf
for (i=0; i <num dsd;="" i++)="" td="" {<=""><td></td><td></td></num>		
reserved	7	uimsbf
connected flag	1	uimsbf
delivery_system_descriptor_tag	8	uimsbf
<pre>if (delivery_system_descriptor_tag == 0x7f) {</pre>		
descriptor tag extension	8	uimsbf
}		
else {		
reserved	8	uimsbf
}		
}		
}		

tuner status reply tag: This tag shall be set to the value of 0x9F840B.

IP_tune_capable_flag: This flag shall be set to 0b1 if the Host is able to accept *tune_ip_req()* APDUs for IP-delivered services and shall be set to 0b0 if the Host is not able to do so.

num_dsd: This field specifies the number of used DSD types supported by the Host. Where one physical tuner supports multiple DSDs, all DSDs shall be reported.

connected_flag: This field specifies if the host believes that at least one tuner of this dsd type is connected to a broadcast network. This is only a hint to the CICAM if a tune request would be successful. A value of 0b0 indicates that the dsd is not connected; a value of 0b1 indicates that the dsd is connected.

delivery_system_descriptor_tag: This field specifies the delivery systems which the Host supports and are available for background tunes. Tag values are defined in DVB SI specification [10].

descriptor_tag_extension: This field, along with the *delivery_system_descriptor_tag*, specifies the delivery systems supported by the Host and being available for background tunes. Tag values are defined in the DVB SI specification [10].

14 CICAM Virtual Channel

14.1 Introduction

This clause specifies the mechanism by which the user is able to select to run an application or high-level MMI session offered by the CICAM. The application or high-level MMI session is offered as a Virtual Channel entry in the Host's channel line-up. When the Virtual Channel is selected by the user, the CICAM application or high-level MMI session is launched.

Application Information resource version 3 and below already contains an APDU to request the launch of an MMI session from the CICAM, but the present document adds a new APDU in order to launch specifically the CICAM Virtual Channel provided to the Host to be entered into its channel list. The Application Information resource thus moves to version 4. The extension and its usage are specified in clause 14.2.

The CICAM informs the Host about its Virtual Channel by including it in the CICAM NIT, using the new version of Operator Profile specified in clause 15.

14.2 Application Information version 4

14.2.1 General

Application Information resource (resource ID 0x00020044) version 4 adds a new command (APDU), namely the enter_cicam_channel() APDU, that requests the launch of the application associated with the CICAM Virtual Channel.

Since the Virtual Channel does not have any associated content in a TS, the usual mechanism for the Host to notify the CICAM of its selection by the user, namely by sending the CA_PMT for the selected service, is not applicable. The Host shall not send the CA_PMT when the CICAM Virtual Channel is selected by the user, and the CICAM shall not expect a CA_PMT or TS from the Host when the Virtual Channel has been selected.

Upon reception of the enter_cicam_channel() APDU the CICAM shall request an Application MMI or High Level MMI session. The Application MMI session shall match the application domain identifier associated with the CICAM's Virtual Channel defined during the Operator Profile installation. The Host shall guarantee that such a session is able to be created when it sends the *enter_cicam_channel()*APDU.

The Host shall send the enter_cicam_channel() APDU only when the CICAM Virtual Channel is to be presented to the user.

1422 enter_cicam_channel APDU

The syntax of the enter_cicam_channel()APDU is shown in table 103. On reception of this APDU, the CICAM shall stop descrambling the Local TS indicated by the LTS_id parameter that it may still be receiving. In single-stream mode the *LTS_id* parameter shall be ignored.

Table 103: enter_cicam_channel APDU syntax

Syntax	Number of bits	Mnemonic
enter_cicam_channel() {		
enter_cicam_channel_tag	24	uimsbf
length_field()= 1		
LTS_id	8	uimsbf
}		

The fields are defined as follows:

enter_cicam_channel_tag: The tag value shall be set to 0x9F8025.

length_field: Length of APDU payload in ASN.1 BER format as defined in EN 50221 [1], clause 8.3.1.

LTS id: Local TS identifier.

Operator Profile version 2 15

15.1 Introduction

Operator Profile Version 2 (with resource ID 0x008F1002) adds the following new features:

- A new profile type which enables the Host to construct a combined logical channel list from the received broadcast SI and additional services provided by the CICAM, signalled in the extended CICAM NIT.
- The ability for the CICAM to add IP delivered services to the Host logical channel list via the addition of the uri_linkage_descriptor in the first loop of the extended CICAM NIT, as specified in clause 15.4.2. The uri_linkage_descriptor carries a URI which links to an Online SDT (OSDT) which describes the IP delivered services.

• The ability for the CICAM to add a Virtual Channel to the Host logical channel list. The CICAM Virtual Channel is specified in clause 14. The attributes of the CICAM Virtual Channel are an LCN, service name and event information. These attributes are communicated in the CICAM Virtual Channel descriptor specified in clause 15.3, which is carried in the extended CICAM NIT.

15.2 Profile Type 2

15.2.1 Introduction

In addition to $profile_type = 0$ and $profile_type = 1$ specified in CI PlusTM V1.3 [3], the new $profile_type = 2$ is specified as follows:

• profile_type = 2 - Profiled operation where the Host constructs a logical channel list from the broadcast SI and the CICAM NIT. The broadcast SI takes precedence over the CICAM NIT. The Host communicates any collisions between broadcast and CICAM NIT, allowing the CICAM to provide an updated CICAM NIT resolving these collisions. This profile allows the CICAM to collect the entitlement rights information. This profile_type allows the CICAM NIT to carry services delivered over IP as well as the CICAM Virtual Channel.

15.2.2 Initialisation and profile discovery

A CICAM that reports *profile_type* = 2 supports a partial channel list to be integrated with a channel list constructed from the broadcast SI. The Host is expected to install the broadcast channel list before installing the operator profile channel lists. The Host then informs of any collisions in requested logical channel numbers (LCN) with the CICAM so that the CICAM is able to provide alternative LCN(s) for the operator profile channels.

The Host shall open a session to the operator profile resource when the CICAM requests a session. The CICAM shall send the *operator_status()* APDU to notify the Host of the operator profile information.

The CICAM shall retain the profile information as defined in CI PlusTM V1.3 [3], clause 14.7.4.1.2.

Profile discovery for a *profile_type* of type 2 adopts the same mechanism as for *profile_type* of type 1. The CICAM may require to scan in order to initialise the profile and build the CICAM NIT.

If the Host supports the delivery systems reported via the *operator_status()* APDU it shall determine the *profile_type* as two (2) and shall install the operator profile channels into the Host logical channel list associated with the delivery system as given in the delivery system hint, at the earliest time possible with minimal disruption to the user. The CICAM shall ensure that the operator profile service list allows the unique identification of each service in the list via its LCN, also in the case when multiple systems are signalled in the system hint, due to the list containing services from several delivery system sources. The *operator_status()* APDU is extended in the present document to add the possibility to signal IP-delivered services in the Operator Profile type 2, as specified in clause 15.4.1.

The Host may decline to store the operator profile if the operator profile delivery descriptor is different to the preselected broadcast service delivery descriptor.

The Host shall store the CICAM Virtual Channel if the application environment for the channel is supported by the Host middleware. If the Host does not support the requested application environment, it shall inform the CICAM of this, so that the CICAM may try to select a different application environment for the Virtual Channel.

If the Host supports DRM descrambling by the CICAM (IP delivery Host player mode) or LSC hybrid connection (IP delivery CICAM player mode), then it shall store the list of available IP-delivered services as defined in clause 15.4.2.

15.2.3 Moving between profiles

There is no facility to move into or out of *profile type* of type 2, so the Host does not need to inform the CICAM that it is in this profile.

15.2.4 LCN collision management

On reception of the CICAM NIT, the Host shall add the operator profile services to the logical channel list. If the Host sees two LCN which are not unique, the Host shall keep the broadcast SI service in its logical channel list. The Host shall inform the CICAM which LCN and service from the CICAM NIT is not unique to the broadcast list, while also providing the closest available free LCNs. The CICAM shall update the CICAM NIT with a new version and shall allocate a new LCN for the service in question.

At any point when the Host detects a collision when services are moved, for example with automatic service updates, the Host shall update its logical channel list according to the broadcast SI. The Host shall inform the CICAM which LCN and service from the CICAM NIT are no longer unique in the broadcast list, while also providing the closest available free LCNs.

The Host shall use the *operator_nit_management()* APDU to inform the CICAM of any LCN which is not unique in the CICAM NIT for a CICAM working in a *profile_type* of type 2.

15.3 CICAM Virtual Channel

15.3.1 CICAM Virtual Channel descriptor

The CICAM Virtual Channel descriptor defined in table 104 shall be present in the first loop of the CICAM NIT and shall be followed by the CI PlusTM private data specifier. The CICAM Virtual Channel may be interpreted by the Host as a data broadcast service. The descriptor provides a LCN, service name, event metadata, and application domain identifier for the service. The CICAM NIT shall not contain more than one Virtual Channel descriptor.

Syntax	Number of bits	Mnemonic
cicam_virtual_channel_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
reserved	2	uimsbf
logical_channel_number	14	uimsbf
service_provider_name_length	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
service_provider_name_char	8	uimsbf
}		
service_name_length	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td></td><td></td></n;>		
service_name_char	8	uimsbf
}		
event_information_length	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td>_</td><td></td></n;>	_	
event_information_char	8	uimsbf
}		
AppDomainIdentifierLength	8	uimsbf
for (i=0; i <n; i++)="" td="" {<=""><td>_</td><td></td></n;>	_	
AppDomainIdentifier	8	bslbf
}		
}		

Table 104: CICAM Virtual Channel descriptor syntax

The fields are defined as follows:

descriptor_tag: This 8-bit field shall be assigned the value 0xCF and shall only be interpreted in the context of a CI PlusTM private data specifier descriptor.

logical_channel_number: This 14-bit field indicates the logical channel number to be assigned to the service, however only 4-digit channel numbers from 0 to 9999 are allowed. Logical channel numbers shall be unique throughout the CICAM NIT i.e. the logical channel number shall be used once only.

service_provider_name_length: This 8-bit field specifies the number of bytes that follow the service_provider_name_length field for describing characters of the name of the service provider.

service_provider_name_char, service_name_char, event_information_char: These are a series of 8-bit character fields. A string of character fields specify the name of the service provider, service, or event information. Text information is coded using the character sets and methods described in EN 300 468 [10], annex A in conjunction with the *operator_info()* APDU.

service_name_length: This 8-bit field specifies the number of bytes that follow the *service_name_length* field for describing characters of the name of the service.

event_information_length: This 8-bit field specifies the length in bytes of the item text which may be used to populate the EPG information field.

AppDomainIdentifierLength: This 8-bit field specifies the length of the string of bytes that specifies the application domain.

AppDomainIdentifier: These bytes specify the required application domain in an application domain specific way. The Application Domain identifier is the same as used by the Application MMI resource.

If the Host does not support the application domain identifier requested by the CICAM, it shall inform the CICAM using the <code>operator_nit_management()</code> APDU to enable the CICAM to select a different application environment for the channel. The CICAM shall inform the Host of the new application environment by providing a new CICAM NIT, with updated NIT version.

If the CICAM only requires High Level MMI for the CICAM Virtual Channel, the CICAM shall report the application domain identifier as the CI PlusTM Browser to ensure that the service is stored on the Host.

15.3.2 Selection of the CICAM Virtual Channel

The Host shall use the Application Information resource version 4, as specified in clause 14, to inform the CICAM that the CICAM Virtual Channel has been selected by the user.

15.3.3 Operator NIT management APDU

The <code>operator_nit_management()</code> APDU shall be sent by the Host to inform the CICAM when there is an issue with the CICAM NIT. On reception of this APDU, the CICAM may update the CICAM NIT to resolve any issue reported by the Host. If the CICAM does not provide an updated CICAM NIT, the Host shall not store services that have an issue in the logical channel list.

If this APDU is not sent by the Host in response to the CICAM sending the <code>operator_nit()</code> APDU, the CICAM shall assume that all LCN requested were available and that the Host supports the indicated application domain.

Table 105: Operator NIT management APDU syntax

Syntax	Number of bits	Mnemonic
<pre>operator_nit_management() {</pre>		
operator_nit_management_tag	24	uimsbf
length_field()		
error_field	8	uimsbf
number_of_services	8	uimsbf
<pre>for (i=0; i<number_of_services; i++)="" pre="" {<=""></number_of_services;></pre>		
channel_issue_type	2	uimsbf
logical_channel_number	14	uimsbf
service_name_length	8	uimsbf
<pre>for (i=0; i<service_name_length; i++)="" pre="" {<=""></service_name_length;></pre>		
char	8	uimsbf
}		
<pre>if (channel_issue_type == 0) {</pre>		
free_lcn_number	8	uimsbf
for (i=0; i <free_lcn_number; i++)="" td="" {<=""><td></td><td></td></free_lcn_number;>		
reserved	2	uimsbf
logical_channel_number	14	uimsbf
}		
}		
}		
]}		

The fields are defined as follows:

operator_nit_management_tag: The tag shall be set to the value 0x9F9C0F.

error_field: This 8-bit field indicates an issue with the CICAM NIT. The allowed values and their meanings are defined in table 106.

Value	Description							
0x00	No issue - OSDT and/or NIT retrieved.							
0x01	NIT error - the NIT cannot be applied.							
	No services from the CICAM NIT have been added to the Host							
	logical channel list.							
0x02	OSDT error - the URL for the OSDT cannot be found.							
	No services from the OSDT have been added to the Host							
	logical channel list, but service from the CICAM NIT will have							
	been added unless indicated otherwise by the							
	operator_nit_management() APDU							
0x03	OSDT error - not supported.							
0x04-0xFF	Reserved for future use.							

Table 106: Error field values

number_of_services: This is an 8-bit field which identifies the number of channels from the CICAM NIT which have not been able to be stored in the Host logical channel list due to some issue detailed by the *channel_issue_type* field.

service_name_length: This 8-bit field specifies the number of bytes that follow the *service_name_length* field for the characters of the service name. The service name shall be the same as that provided by the CICAM in the APDU to which the *operator_nit_management()* APDU corresponds. The origin of the service name can be one of:

- the service name provided in the ciplus_service_descrptor in the CICAM NIT second loop;
- the service name provided in the cicam_virtual_channel_descriptor in the CICAM NIT first loop;

the first service name found in the list of possible service names of the service in the OSDT pointed to in the CICAM NIT first loop.logical_channel_number: This 14-bit field indicates the logical channel number assigned to the service (by the CICAM NIT) that has not been stored in the Host logical channel list.

channel_issue_type: This 2-bit field specifies the issue with the service from the CICAM NIT. The values are defined in table 107.

Value	Description
0x0	LCN collision - denotes that the LCN provided by the CICAM NIT
	is not unique and has a collision with a LCN provided by
	broadcast Service Information.
0x1	Application Domain - denotes that the Application Domain
	requested in the CICAM NIT is not supported by the Host.
0x2	Service not stored - Delivery mechanism not supported.
0x3	Reserved for future use.

Table 107: Channel issue type values

free_lcn_number: This 8-bit field specifies the number of free LCN values where the service may be moved without causing another collision. The Host shall at a minimum provide the first available LCN below and above the requested LCN from the CICAM NIT, if available.

15.4 IP delivered services

15.4.1 operator_status() APDU

The *operator_status()* APDU is extended to add the option to the *delivery_system_hint* field, using the reserved bit in Operator Profile version 1, as defined in table 14.42 in the CI PlusTM V1.3 specification [3]. Table 108 shows the set of *delivery_system_hint* values for Operator Profile version 2.

Table 108: delivery_system_hint values

Bit	Description							
0b0001	This is a cable network and may be DVB-C and/or DVB-C2							
0b0010	This is a satellite network and may be DVB-S and/or DVB-S2							
0b0100	This is a terrestrial network and may be DVB-T and/or DVB-T2							
0b1000	This is an IP network.							

15.4.2 IP service discovery and update

For IP delivered services, the CICAM NIT may include one or more *uri_linkage_descriptors* in the first loop of the CICAM NIT. The *uri_linkage_descriptor* is defined in the DVB SI specification [10], clause 6.4.14.

Only *uri_linkage_descriptors* containing a *uri_linkage_type* with value 0x00 are defined for use within this scope. The *uri_linkage_descriptor* carries a single URI which provides the location of the list of IP delivered services. This list is declared in an XML data structure, called the Online SDT (OSDT), which contains a single *IPServiceList* element. The OSDT XML schema is specified in annex D. The full normative XML schema is available as the file osdt_v1.1.1.xsd in archive ts_103205v010101p0.zip, which accompanies the present document. A future revision of the present document may refer to an external specification for the syntax of the OSDT.

Services listed in the OSDT may include:

- Services for which service location information is provided but none of these locations is supported by the Host or the CICAM. The Host may not add these services to the channel list.
- Services for which service location information is provided but the Host does not support any of the service locations provided while the CICAM supports at least one of them. The Host determines that the CICAM supports a service location by using the CICAM_player_verify_req() APDU. In this case, when the service is selected, the Host shall initiate the service retrieval using the CICAM_player_play_req() APDU (see clause 8.15.9).
- Services for which at least one service location is provided and the Host supports at least one of them. When the service is selected, the Host shall retrieve the service. It may use Sample Mode to pass the content to the CICAM if the service is encrypted (see clause 7.2).
- Services for which no service location information is provided. If application location information is provided, these are data services (unless the ServiceType indicates otherwise). When the service is selected, the Host shall attempt to launch the application (how this is done is outside the scope of the present document).

If the URI in the <code>uri_linkage_descriptor</code> is "cicam:", the Host shall use the <code>operator_osdt_request()</code> APDU to request the OSDT from the CICAM. In this case, the <code>min_polling_interval</code> field in the <code>uri_linkage_descriptor</code> shall be ignored by the Host. If the URI defines a different scheme to the one above (for example "http://www.example.com/path/list.xml"), the Host shall retrieve the OSDT using the specified URI and shall check for any updates to the OSDT not more frequently than the polling interval indicated in the <code>min_polling_interval</code> field. URIs using scheme other than "http:" or "https:" may be ignored by the Host.

For each OSDT file, the Host should add all of the listed IP services to the channel line up in the same way that it does for any other services referenced from the CICAM NIT. This may include discarding services which are not compatible with the Host, and internally resolving any LCN conflicts.

Whenever the CICAM advertises a new version of the CICAM_NIT, the Host shall retrieve the OSDT, whatever its location, if it has already stored IP-delivered services provided by the CICAM.

15.4.3 operator_osdt_request APDU

The Host sends this APDU to the CICAM to query the current OSDT file. Its syntax is shown in table 109. The CICAM shall reply with an *operator_osdt_reply()* APDU returning the OSDT to the Host.

Table 109: operator_osdt_request APDU syntax

Syntax	No. of bits	Mnemonic
<pre>operator_osdt_request() { operator_osdt_request_tag length_field() }</pre>	24	uimsbf

The fields are defined as follows:

operator_osdt_request_tag: The tag shall be set to the value 0x9F9C0D.

15.4.4 operator_osdt_reply APDU

The CICAM shall send this APDU to the Host in response to an *operator_osdt_request()* APDU. Its syntax is shown in table 110. The APDU, if successful, contains the latest OSDT file.

Table 110: operator_osdt_reply APDU syntax

Syntax	No. of bits	Mnemonic
operator_osdt_reply () {		
operator_osdt_reply_tag	24	uimsbf
length_field()		
osdt_file_data_length	32	uimsbf
<pre>for (i=0; i< osdt_file_data_length; i++) {</pre>		
osdt file data byte	8	uimsbf
] } = = = = =		
}		

The fields are defined as follows:

operator_osdt_reply_tag: The tag shall be set to the value 0x9F9C0E.

file_data_data_length: The length of the OSDT file in bytes. If the CICAM has no OSDT to return, then the $osdt_file_data_length$ shall be set to zero.

osdt_file_data_byte: A byte of the OSDT XML file.

16 High-Level MMI

A general recommendation is that High-Level MMI should be implemented on the Host in a way so as to avoid disruption to applications that may be running on the Host.

The following additional requirements are specified, which clarify the Host minimum capabilities relating to High-Level MMI, as specified in EN 50221 [1] clause 8.6:

- The Host shall be able to display a minimum of 40 characters per line (text item, title, sub-title, bottom line).
- The Host shall be able to display a *menu()* object containing up to 254 items.
- The Host shall be able to display a *list()* object containing up to 254 items.
- When presenting a *menu()* or *list()* object, the Host shall be capable of displaying a minimum of 5 items simultaneously, and shall implement scrolling of the object, if not all items fit on the displayed area.

Annex A (normative): Resources summary

Table A.1 summarizes the complete set of resources for the present document, consisting of resources defined in previous specifications and the new resources and resource types that are defined in the present document.

Table A.1: Summary of CI Plus™ resources

	Reso	ource			Application C		Direc		
Name	Resource Identifier	class	type	Vers.	APDU Tag	Tag value	Host	CIC AM	Spec
	00 01 00 41				profile_enq	9F 80 10	← ·		
		1	1	1	profile	9F 80 11	←		EN 50221 [1]
_					profile_change	9F 80 12	←		
Resource					profile_enq	9F 80 10	← ·		-
Manager	00 01 00 42 1	1	1	2	profile	9F 80 11 9F 80 12	← ·		TC 404 600 [0]
		'	'		profile_change CICAM id send	9F 80 12	← ·		TS 101 699 [2]
					CICAM_id_command	9F 80 14)		-
					application_info_enq	9F 80 20)		
	00 02 00 41	2	1	1	application_info	9F 80 21	+		EN 50221 [1]
					enter_menu	9F 80 22)		1
					application_info_enq	9F 80 20)		
	00 02 00 42	2	1	2	application_info	9F 80 21	+		TS 101 699 [2]
					enter_menu	9F 80 22	→		
					application_info_enq	9F 80 20	→		
Application					application_info	9F 80 21	+		
Information	00 02 00 43	2	1	3	enter_menu	9F 80 22)		Cl Plus™ 1.3 [3]
					request_CICAM_reset	9F 80 23	<u> </u>		-
					data_rate_info	9F 80 24)		
					application_info_enq application info	9F 80 20 9F 80 21			-
					enter menu	9F 80 22	-		Present
	00 02 00 44	2	1	4	request_CICAM_reset	9F 80 23	<i>+</i>		document
					data rate info	9F 80 24)		
					enter_cicam_channel	9F 80 25)		
0 1111					ca_info_enq	9F 80 30	-		
	00 00 00 44	0			ca_info	9F 80 31	+	ı	EN 50004 [1]
	00 03 00 41	3	1	1	ca_pmt	9F 80 32	→		EN 50221 [1]
Conditional Access					ca_pmt_reply	9F 80 33	+		
Support	00 03 00 81				ca_info_enq	9F 80 30	→		
	(see note 1)	3	2	1	ca_info	9F 80 31	+		Present
	(000		_	-	ca_pmt	9F 80 32)		document
					ca_pmt_reply	9F 80 33	+		
		32			tune	9F 84 00	←		_
	00 20 00 41		1	1	replace	9F 84 01 9F 84 02	-		EN 50221 [1]
					clear_replace ask release	9F 84 03	-		-
					tune	9F 84 00	<i>+</i>		
					replace	9F 84 01	+		
					clear_replace	9F 84 02	+		
	00 20 00 42	32	1	2	ask_release	9F 84 03)		CI Plus™ 1.3 [3]
					tune_broadcast_req	9F 84 04	+		
					tune_reply	9F 84 05	→		
					ask_release_reply	9F 84 06	+		
					tune	9F 84 00	+		
					replace	9F 84 01	+		4
					clear_replace	9F 84 02	<u> </u>		_
Host Control					ask_release	9F 84 03)		-
					tune_broadcast_req	9F 84 04	+		Donoros
	00 20 00 43	32	1	3	tune_reply ask_release_reply	9F 84 05 9F 84 06)		Present document
					tune_lcn_req	9F 84 06	+		- GOCGINEIIL
					tune_icn_req	9F 84 08	-		1
					tune_triplet_req	9F 84 09	+		- - -
					tune_status_req	9F 84 0A	+		
					tune_status_reply	9F 84 0B	-		
					ask_release	9F 84 03)		
					tune_broadcast_req	9F 84 04	+		
	00 20 00 81				tune_reply	9F 84 05	→		Present
	(see note 1)	32	2	1	ask_release_reply	9F 84 06	+		- document
					tune_lcn_req	9F 84 07	+		document
					tune_ip_req	9F 84 08	+		4
		1		1	tune_triplet_req	9F 84 09	+		

	Reso	Resource			Application Ob		Direc													
Name	Resource Identifier	class	type	Vers.	APDU Tag	Tag value	Host	CIC AM	Spec											
					tune_status_req	9F 84 0A	+													
					tune_status_reply	9F 84 0B 9F 84 40	→ ←													
Date-Time	00 24 00 41	36	1	1	date_time_enq date_time	9F 84 41	→		EN 50221 [1]											
					close_mmi	9F 88 00	<i>,</i>													
					display_control	9F 88 01	-													
					display_reply	9F 88 02)													
					text_last	9F 88 03	+													
					text_more	9F 88 04	+													
					keypad_control	9F 88 05	<u> </u>													
					keypress enq	9F 88 06 9F 88 07	→ ←													
					answ	9F 88 08	→													
					menu_last	9F 88 09	/													
					menu_more	9F 88 0A	-													
	00 40 00 41	64	1	1	menu_answ	9F 88 0B	→	,	EN 50004 [1]											
	00 40 00 41	64	1	1	list_last	9F 88 0C	+	•	EN 50221 [1]											
					list_more	9F 88 0D	+													
					subtitle_segment_last	9F 88 0E	+													
					subtitle_segment_more	9F 88 0F	→													
MMI					display_message	9F 88 10	+													
					scene_end_mark scene done	9F 88 11 9F 88 12	←													
					scene_done	9F 88 13	→													
					subtitle_download_last	9F 88 14	′													
					subtitle_download_more	9F 88 15	· ·													
					flush_download	9F 88 16	+													
					download_reply	9F 88 17	+	•												
								close_mmi	9F 88 00	→										
					display_control	9F 88 01	+													
					display_reply	9F 88 02	→													
				1	enq	9F 88 07	<u> </u>													
	00 40 00 81 (see note 1)	64	2		answ manu loot	9F 88 08	→ ←		Present											
	(See Hote 1)				menu_last menu_more	9F 88 09 9F 88 0A	-		document											
					menu_answ	9F 88 0B	<u> </u>		4											
					list_last	9F 88 0C	,													
					list_more	9F 88 0D	+		1											
low speed					comms_cmd	9F 8C 00	+	•	EN 50221 [1]											
comms.					connection_descriptor	9F 8C 01	+													
					comms_reply	9F 8C 02	\rightarrow													
	00 60 xx x1	96		1	comms_send_last	9F 8C 03	+													
					comms_send_more	9F 8C 04	<u> </u>		- - -											
					comms_rcv_last	9F 8C 05 9F 8C 06	→ →													
					comms_rcv_more comms_cmd	9F 8C 00	→ → →													
					connection_descriptor	9F 8C 01	· +													
					comms_reply	9F 8C 02	`													
	00 60 xx x2	96		2	comms_send_last	9F 8C 03	+		CI Plus™ 1.3 [3]											
					comms_send_more	9F 8C 04	+]											
					comms_rcv_last	9F 8C 05	\rightarrow													
					comms_rcv_more	9F 8C 06	→													
				1	comms_cmd	9F 8C 00	-		4											
					connection_descriptor	9F 8C 01 9F 8C 02	←		-											
	00 60 xx x3	96		3	comms_reply comms_send_last	9F 8C 02 9F 8C 03	-		Cl Plus™ 1.3 [3]											
	00 00 11 10				comms_send_more	9F 8C 04	`		_[C] [[]											
				1	comms_rcv_last	9F 8C 05	→		1											
		<u> </u>	<u> </u>	<u> </u>	comms_rcv_more	9F 8C 06	→													
					comms_cmd	9F 8C 00	+													
				1	connection_descriptor	9F 8C 01	+													
				1	comms_reply	9F 8C 02)		4											
				1	comms_send_last	9F 8C 03	+		4											
	00 60 xx x4	96		4	comms_send_more	9F 8C 04	←		Present											
	00 00 XX X4	90		4	comms_rcv_last comms_rcv_more	9F 8C 05 9F 8C 06	<u> </u>		document											
				1	comms_info_req	9F 8C 07			1											
				1	comms_info_reply	9F 8C 08	→		1											
				1	comms_IP_config_req	9F 8C 09	/		1											
				1	comms_IP_config_reply	9F 8C 0A	→		1											
					cc_open_req	9F 90 01	+													
				1	cc_open_cnf	9F 90 02	→													
Content				1	cc_data_req	9F 90 03	+													
Control	00 8C 10 01	140	64	1	cc_data_cnf	9F 90 04	→		CI Plus™ 1.3 [3]											
					C	cc					C	C	C	'	'	cc_sync_req	9F 90 05	<u> </u>		4
				1	cc_sync_cnf	9F 90 06)		4											
		1	<u> </u>	1	cc_sac_data_req	9F 90 07	+	•	<u> </u>											

	Reso	ource	•	•	Application Ob		Direc			
Name	Resource Identifier	class	type	Vers.	APDU Tag	Tag value	Host	CIC	Spec	
					cc_sac_data_cnf	9F 90 08	→			
					cc_sac_sync_req	9F 90 09	+			
					cc_sac_sync_cnf	9F 90 10	→			
					cc_open_req	9F 90 01	+			
					cc_open_cnf	9F 90 02	→			
					cc_data_req	9F 90 03	<u> </u>		4	
					cc_data_cnf	9F 90 04	<u>→</u>		4	
					cc_sync_req	9F 90 05	+		4	
					cc_sync_cnf	9F 90 06	→ ←			
					cc_sac_data_req cc_sac_data_cnf	9F 90 07 9F 90 08			4	
	00 8C 10 02	140	64	2	cc_sac_data_crii cc_sac_sync_req	9F 90 09			_ CI Plus™ 1.3 [3]	
	00 00 10 02	140	04		cc_sac_sync_cnf	9F 90 10	\ \ \ \ \		OI Flus : " 1.3 [J]	
					cc_PIN_capabilities_req	9F 90 11	/		=	
					cc_PIN_capabilities_reply	9F 90 12	+			
				cc PIN cmd	9F 90 13	<u>`</u>		1		
					cc_PIN_reply	9F 90 14	+			
					cc_PIN_event	9F 90 15	+			
					cc_PIN_playback	9F 90 16	→			
					cc_PIN_MMI_req	9F 90 17	→	,		
			İ		cc_open_req	9F 90 01	-			
			1		cc_open_cnf	9F 90 02	→]	
					cc_data_req	9F 90 03	+			
					cc_data_cnf	9F 90 04	→			
			1		cc_sync_req	9F 90 05	+			
			1		cc_sync_cnf	9F 90 06	→			
					cc_sac_data_req	9F 90 07	+			
					cc_sac_data_cnf	9F 90 08	→		Present	
	00 8C 10 03	140	64	3	cc_sac_sync_req	9F 90 09	+		document	
					cc_sac_sync_cnf	9F 90 10	\rightarrow		document	
						cc_PIN_capabilities_req	9F 90 11	\rightarrow		
					cc_PIN_capabilities_reply	9F 90 12	+			
					cc_PIN_cmd	9F 90 13	→			
					cc_PIN_reply	9F 90 14	+			
					cc_PIN_event	9F 90 15	+		4	
					cc_PIN_playback	9F 90 16	→		4	
					cc_PIN_MMI_req	9F 90 17	<u>→</u>			
					cc_open_req	9F 90 01	<u> </u>		4	
					cc_open_cnf	9F 90 02)			
					cc_data_req	9F 90 03	<u>+</u>		=	
					cc_data_cnf	9F 90 04	→ ←			
					cc_sync_req	9F 90 05				
					cc_sync_cnf cc_sac_data_req	9F 90 06 9F 90 07	→ ←		4	
					cc_sac_data_req	9F 90 08	→		+	
	00 8C 10 41	140	65	1	cc_sac_data_crii cc_sac_sync_req	9F 90 09			Present	
	(see note 1)	140	0.5	'	cc_sac_sync_cnf	9F 90 10	<u>`</u>		document	
					cc_PIN_capabilities_req	9F 90 11	,		1	
					cc_PIN_capabilities_reply	9F 90 12	+			
					cc_PIN_cmd	9F 90 13	→			
					cc_PIN_reply	9F 90 14	-		1	
			1		cc_PIN_event	9F 90 15	+		1	
					cc_PIN_playback	9F 90 16	→	·		
		<u> </u>	<u></u>		cc_PIN_MMI_req	9F 90 17	→			
Jost					Host_country_enq	9F 81 00	+			
Host ∟anguage &	00 8D 10 01	141	64	1	Host_country	9F 81 01	→	,	 Cl Plus™ 1.3 [3]	
Country	00 00 10 01	'-'	0-	'	Host_language_enq	9F 81 10	+		OI FIUS 1.3 [3]	
,				<u> </u>	Host_language	9F 81 11	→			
			1		cicam_firmware_upgrade	9F 9D 01	+	•	4	
					cicam_firmware_upgrade_	9F 9D 02	→	•		
CICAM_Upg	00.05.40.04	4.40	0.4	_	reply	+ 	<u> </u>			
ade	00 8E 10 01	142	64	1	cicam_firmware_upgrade_	9F 9D 03	+		CI Plus™ 1.3 [3]	
			1		progress	-	+		4	
			1		cicam_firmware_upgrade_ complete	9F 9D 04	+			
		+	†	1	operator_status_req	9F 9C 00	→	•		
			1		operator_status	9F 9C 01			1	
					operator_nit_req	9F 9C 02	\ \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\		1	
			1		operator_nit	9F 9C 03	/		1	
			1		operator_info_req	9F 9C 04	→		1	
Operator			1		operator_info	9F 9C 04	 		1	
Profile	00 8F 10 01	143	64	1	operator_search_start	9F 9C 06	<u>`</u>		Cl Plus™ 1.3 [3]	
. 55			1		operator_search_status	9F 9C 07			1	
			1		operator_search_status	9F 9C 08	\rightarrow		1	
			1		operator_tune	9F 9C 09			1	
			1		operator_tune_status	9F 9C 0A	→ ×		1	
	1	1	1	Ì	operator_entitlement_ack	9F 9C 0B	<i>'</i>		≟	

Resource			Application Object		Direction						
Name	Resource Identifier	class	type	Vers.	APDU Tag	Tag value	Host	CIC AM	Spec		
					operator_search_cancel	9F 9C 0C	→				
					operator_status_req	9F 9C 00)				
					operator_status	9F 9C 01 9F 9C 02	←		1		
					operator_nit_req operator_nit	9F 9C 02	-		1		
					operator_info_req	9F 9C 04	→		-		
					operator_info	9F 9C 05	<i>+</i>				
					operator_search_start	9F 9C 06	<u>`</u>		1		
					operator search status	9F 9C 07	+		Present		
	00 8F 10 02	143	64	2	operator_exit	9F 9C 08	→		document		
					operator_tune	9F 9C 09	+	-			
					operator_tune_status	9F 9C 0A	→				
					operator_entitlement_ack	9F 9C 0B	→	,			
					operator_search_cancel	9F 9C 0C	→				
					operator_osdt_request	9F 9C 0D	\rightarrow				
					operator_osdt_reply	9F 9C 0E	+				
					operator_nit_management	9F 9C 0F	→				
					SAS_connect_rqst	9F 9A 00)				
					SAS_connect_cnf	9F 9A 01	+				
					SAS_data_rqst (see note	9F 9A 02	←-	→			
					2) SAS_data_av (see note 2)	9F 9A 03		$\overline{}$	4		
SAS	00 96 10 01	150	64	1	SAS_data_av (see note 2)	9F 9A 03	← -		CI Plus™ 1.3 [3]		
343	00 90 10 01	130	04	'	SAS_data_crif (see flote 2) SAS_server_query (see				[C] Pius''' 1.3 [3]		
					note 2)	9F 9A 05	← -	}			
					SAS_server_reply (see note 2)	9F 9A 06	← -				
					SAS_async_msg	9F 9A 07	(-				
					RequestStart	9F 80 00	<u> </u>		_		
					RequestStartAck	9F 80 01	→ →		4		
	00 41 00 41	65	1	1	FileRequest FileAcknowledge	9F 80 02	→		TS 101 699 [2]		
					AppAbortRequest	9F 80 03 9F 80 04	(-		-		
					AppAbortAck	9F 80 05	-		-		
					RequestStart	9F 80 00	`				
			65 1		RequestStartAck	9F 80 01	· ·		CI Plus™ 1.3 [3]		
	00 41 00 42	65		2	FileRequest	9F 80 02	,				
					FileAcknowledge	9F 80 03	· ·				
					AppAbortRequest	9F 80 04	(-	>			
Application					AppAbortAck	9F 80 05	← -				
MMI					RequestStart	9F 80 00	+	-			
					RequestStartAck	9F 80 01	→	,	1		
	00 41 00 43	FileRec	C.F.	FileRequest	9F 80 02	→	,	Present			
	00 41 00 43	65	65 1 3	1 3	1 3	5 1	FileAcknowledge	9F 80 03	+		document
					AppAbortRequest	9F 80 04	(-)	-		
					AppAbortAck	9F 80 05	←-				
					RequestStart	9F 80 00	+				
					RequestStartAck	9F 80 01	→				
	0 41 00 81 (see	65	2 1	FileRequest	9F 80 02	→		Present			
	note 1)		_		FileAcknowledge	9F 80 03	· ·		document		
					AppAbortRequest	9F 80 04	← ·				
					AppAbortAck	9F 80 05	←-	/			
	00 90 00 41			1	CICAM_multistream_ capability	9F 92 00	-		Present		
Multi-stream	(see note 1)	144	1	1 1	PID select req	9F 92 01	+		document		
	(333 110.0 1)				PID_select_reply	9F 92 02	<u>`</u>		-		
				†	FileSystemOffer	9F 94 00	<i>+</i>				
Auxiliary File	00.04.00.44	4.45	_		FileSystemAck	9F 94 01	→		Present		
System	00 91 00 41	145	1	1	FileRequest	9F 94 02	→		document		
	<u> </u>	<u> </u>		<u> </u>	FileAcknowledge	9F 94 03	+		<u> </u>		
					sd_info_req	9F 98 00	→				
				1	sd_info_reply	9F 98 01	+		_		
Sample	00 92 00 41	146	1	1	sd_start	9F 98 02	\rightarrow		Present		
Decryption	00 02 00 41	146	'	1 '	sd_start_reply	9F 98 03	+		document		
				1	sd_update	9F 98 04	→		1		
				1	sd_update_reply	9F 98 05	+				
				1	CICAM_player_verify_req	9F A0 00)		4		
				1	CICAM_player_verify_reply	9F A0 01	+		4		
				1	CICAM_player_capabilities	9F A0 02	-	-			
CICAM Player				1	_req CICAM_player_capabilities				Present document		
					_reply	9F A0 03	\rightarrow	,			
	00 93 00 41	147	47 1	1	CICAM_player_start_req	9F A0 04	→	,			
. 10,01					CICAM_player_start_reply	9F A0 05	→				
1				1	CICAM_player_play_req	9F A0 06	<i>→</i>		†		
1				1	CICAM_player_status_				1		
1					error	9FA0 07	+	,			
				1	CICAM_player_control_req	9F A0 08	→	,	1		
		•	•	•			•				

Resource				Application Object		Direction			
Name	Resource Identifier	class	type	Vers.	APDU Tag	Tag value	Host	CIC AM	Spec
					CICAM_player_info_req	9F A0 09	->	,	
					CICAM_player_info_reply	9F A0 0A	+	•	
					CICAM_player_stop	9F A0 0B	+)	
					CICAM_player_session_ end	9F A0 0C	+	-	
					CICAM_player_asset_end	9F A0 0D	+		

This resource is applicable for multi-stream operation as specified in clause 6.

The cc_sac_data_req() and cc_sac_data_cnf() APDUs use LTS_id as a datatype field for URI transmission and acknowledgment. NOTE 1: NOTE 2:

Annex B (normative):

Credential Specification: Parameters exchanged in APDUs

Table B.1: APDU parameters summary

Key or variable	Size (bits)	Comments	Datatype id
Reserved	-	-	1
Reserved	-	-	2
Reserved	-	-	3
Reserved	-	-	4
HOST_ID	64	Generated by the ROT and included in the X.509 certificate	5
CICAM_ID	64	Generated by the ROT and included in the X.509 certificate	6
Host_BrandCert	Note 1	Host Brand Certificate	7
CICAM_BrandCert	Note 1	CICAM Brand Certificate	8
Reserved	-	-	9
Reserved	-	-	10
Reserved	-	-	11
Кр	256	CICAM's key precursor to Host for CCK	12
DHPH	2048	DH Public Key Host	13
DHPM	2048	DH Public Key CICAM/CICAM	14
Host_DevCert	Note 1	Host Device Certificate Data	15
CICAM_DevCert	Note 1	CICAM Device Certificate Data	16
Signature_A	2048	The signature of Host DH public key	17
Signature_B	2048	The signature of CICAM DH public key	18
auth_nonce	256	Random nonce of 256 bits generated by the CICAM and	19
_		transmitted by the CICAM to the Host for use in the	
		authentication protocol	
Ns_Host	64	Host's challenge to CICAM for SAC	20
Ns CICAM	64	CICAM's challenge to Host for SAC	21
AKH	256	Authentication Key Host	22
AKM	256	Authentication Key CICAM/CICAM	23
Reserved	-	-	24
uri_message	64	Data message carrying the Usage Rules Information	25
program_number	16	MPEG program number	26
uri_confirm	256	Hash on the data confirmed by the Host	27
key register	8	(uimsbf) 0 = even, 1 = odd, other values not supported	28
uri_versions	256	Bitmask expressing the URI versions that can be supported	29
_		by the Host. Format is ' uimsbf'	
status_field	8	Status field in APDU confirm messages	30
Srm_data_hdcp	variable	SRM for HDCP (see Note 2)	31
Srm_confirm	256	Hash on the data confirmed by the Host	32
CICAM_license	variable	License from CICAM associated with content (Note 3)	33
license status	8	Current status of the content license	34
license_rcvd_status	8	Status from the exchange of content license	35
Host_license	variable	License for which the Host requires current status. (Note 3)	36
play_count	8	Remaining Play Count	37
operating_mode	8	Record operating mode	38
PINcode data	variable	CICAM PIN code one byte for each pin code digit	39
record_start_status	8	CICAM status after a record_start protocol	40
mode_change_status	8	CICAM status after a change operating mode protocol	41
record_stop_status	8	CICAM status after a record_stop protocol	42
srm_data_dtcp	variable	SRM for DTCP (see note 4)	43
Reserved	-	-	44
Reserved	-	-	45
Reserved	_	-	46
	ı	1	

Key or variable	Size (bits)	Comments	Datatype id
Reserved	-	-	47
Reserved	-	-	48
Reserved	-	-	49
LTS_id	8	Local TS Identifier	50

- NOTE 1: Certificate lengths are variable.

 NOTE 2: SRMs for HDCP are defined in the HDCP specification [24]. First generation SRMs shall not exceed
- NOTE 3: Licenses shall not be zero length and shall be padded to the next byte boundary. Licenses shall be no larger than 1 024 bytes.
- NOTE 4: SRMs for DTCP are defined in the DTCP specification [25]. First generation SRMs shall not exceed 5 kilobytes.

Annex C (informative): Descriptor identification and location

Table C.1 lists the descriptors declared or defined within the present document and within previous versions of the CI PlusTM specification, listing the descriptor tag values, the reference of the document where each is defined, and the intended placement within the tables. This does not imply that their use in other tables is restricted.

Table C.1: Descriptor identification and location

Descriptor	Tag Value	Defined in	Possible Locations
ciplus_content_label_descriptor	0xCB	CI Plus™ V1.3 [3] N.1.2.4	CICAM NIT
ciplus_service_descriptor	0xCC	CI Plus™ V1.3 [3]] N.1.2.3	CICAM NIT
ciplus_initialization_vector_descriptor	0xCD	Present document, clause 7.5.5.9.2	SST
ci_protection_descriptor	0xCE	CI Plus™ V1.3 [3]] 10.1.1.1	SDT
ciplus_key_identifier_descriptor	0xCF	Present document, clause 7.5.5.9.3	SST
Reserved	0xD0 to		
	0xEF		
Host defined	0xF0 to		SST, SET, FLT
	0xFE		
Forbidden	0xFF		

Annex D (normative): Schemas

D.1 OSDT schema

D.1.1 General

The Online SDT (OSDT) is an XML structure that contains a list of available IP-delivered services.

Clauses D.1.2 and D.1.3 define the various types and elements that are used in the OSDT XML schema. The full normative XML schema is contained as the file "osdt.xsd" in archive ts_103205v010101p0.zip which accompanies the present document.

Clause D.1.4 provides an informative example of OSDT usage.

D.1.2 Namespace

The namespace of the OSDT schema is urn:dvb:metadata:ciplus:osdt:2013.

D.1.3 Complex types and attribute groups

D.1.3.1 SubRegionType

```
<complexType name="SubRegionType">
   <sequence>
      <element name="Region" type=" mpeg7:TextualType" maxOccurs="unbounded"/>
      <element name="SubRegion" type="dvb:SubRegionType" minOccurs="0"/>
   </sequence>
</complexType>
<complexType name="RegionListType">
   <sequence>
      <element name="PrimaryRegion" type="dvb:SubRegionType" minOccurs="0"/>
   </sequence>
   <attribute name="CountryCodes" type=" sds:ISO-3166-List"/>
</complexType>
<complexType name="TargetRegionType">
   <sequence>
      <element name="RegionList" type="dvb:RegionListType" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="AccessibleOutOfRegion" type="boolean" default="true"/>
</complexType>
```

Element/Attribute	Semantics
SubRegionType	A type used to provide the name of the sub-region.
Region	The name of the sub-region. Multiple elements of this type may be provided as long as they have different languages.
RegionListType	A type used to define the region and provide the region name.
PrimaryRegion	The details of the region, defined in a hierarchical manner starting from the primary region.
CountryCodes	The list of countries that make up the region which is further defined by the PrimaryRegion element.
TargetRegionType	A type used to provide the country and region within the country where the service is intended to be received. Where this is intended to be equivalent to the target region descriptor, the use of sub-regions shall be limited to two levels (i.e. primary regions containing sub-regions which contain sub-regions).
RegionList	The list of regions within the countries.
AccessibleOutOfRegion	A flag indicating whether the service should be accessed when the Host is not in one of the listed regions.

D.1.3.2 ServiceLocationType

```
<complexType name="ServiceLocationType">
   <sequence>
      <any processContents="lax" id="ProtocolDependentInformation" />
      <element name="DRMControlInformation" type="casd:DRMControlInformationType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="ContentAttributes" type="dvb:ContentAttributesType" minOccurs="0"/>
      <choice>
         <element name="IPMulticastAddress" type="sds:McastType"/>
         <element name="RTSPURL" type="sds:RTSPURLType"/>
         <element name="UriBasedLocation" type="dvb:ExtendedURIType"/>
      </choice>
   </sequence>
   <attribute name="priority" type="integer" default="0"/>
</complexType>
<complexType name="ExtendedURIType">
   <sequence>
      <element name="URI" type="anyURI"/>
   <attribute name="contentType" type="mpeg7:mimeType" use="required"/>
</complexType>
<complexType name="ContentAttributesType">
   <sequence>
      <element name="AudioAttributes" type="tva:AudioAttributesType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="VideoAttributes" type="tva:VideoAttributesType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="CaptionLanguage" type="tva:CaptionLanguageType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="SignLanguage" type="tva:SignLanguageType" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
</complexType>
<complexType name="LCNType">
   <attribute name="LCN" type="positiveInteger" use="required"/>
   <attribute name="subscribed" type="boolean" use="optional"/>
   <a tribute name="selectable" type="boolean" use="optional" default="true"/>
   <attribute name="visible" type="boolean" use="optional" default="true"/>
</complexType>
<complexType name="IPServiceType">
   <sequence>
      <element name="UniqueIdentifier" type=" sds:TextualIdentifier"/>
      <element name="DVBTriplet" type="sds:DVBTriplet" minOccurs="0"/>
      <element name="ServiceLocation" type="dvb:ServiceLocationType" minOccurs="0" maxOccurs="unbounded" />
      <element name="LCN" type="dvb:LCNType" minOccurs="0"/>
```

Element/Attribute	Semantics
ServiceLocationType	A type used to provide the location information for the service along
	with DRM information and audio/video information.
DRMControlInformation	Used to provide DRM information, including the DRM system ID and
	other metadata, for this version of the service. The DoNotRecord and
	DoNotTimeShift elements may be regarded by the device as a hint:
	the URI provided by the CICAM shall always take precedence.
ContentAttributes	The attributes of the audio, video, captioning and signing of this version of the service.
IPMulticastAddress	Signals the use of IGMP to access the service and provides the
	transport address and other parameters at which the service
	may be accessed.
RTSPURL	Signals the use of RTSP to access the service and provides the
	URL at which the service description may be accessed.
	This URL is also the aggregate URL when control URLs are
	present for FEC streams.
UriBasedLocation	Provides the URI where the service is located, where the target of the
	URI has the MIME type as provided in the contentType attribute.
priority	The priority of this ServiceLocationType element relative to other
	ServiceLocationType elements for the service.
ExtendedURIType	A type used to provide a URI with additional information. This type
	may be used for ProtocolDependentInformation.
contentType	The MIME type of the object identified by the URI.
URI	The URI providing the location of the service.
ContentAttributesType	A type used to provide the audio, video and other attributes of the
	service.
AudioAttributes	The audio attributes of the service.
VideoAttributes	The video attributes of the service
CaptionLanguage	The language of the captions on the service.
SignLanguage	The language of the signing with the service.
LCNType	A type used to provide the logical channel number for the service.
LCN	The logical channel number. The semantics for this attribute are the
	same as for the logicial_channel_number field in the
	ciplus_service_descriptor (see annex N.1.2.3 of [3]).
subscribed	A flag indicating whether the user has subscribed to this service or
	not. When false, the device can assume that it will not be able to
	present this service. If this attribute is not provided, the subscription
	status is not known.
selectable	A flag indicating whether the device should allow the service to be
	selected via direct numerical entry of the logical channel number.
	This flag is only interpreted when the visible flag is set to false. When
	set to true, the flag indicates that the hidden service is selectable by
	direct entry of the logical channel number; when set to false, then the
	hidden service is not directly selectable by the user (but may be
	selectable by LCN from an application environment).

Element/Attribute	Semantics
visible	A flag indicating whether the device should include this service in any service list or EPG presented to the viewer. When set to true, this flag indicates that the service is normally visible via the Host service or channel list and EPG etc. When set to false, this indicates that the receiver is not expected to offer the service to the user in normal navigation modes but the receiver shall provide a mechanism to access these services by direct entry of the logical channel number, depending on the setting of the selectable flag.
IPServiceType UniqueIdentifier	A type used to provide the details of the service. The unique ID of the service. This ID should never be changed for a
Oniqueidentinei	service, even if all other parameters of the service are changed for a service, even if all other parameters of the service are changed. The child attribute DomainName, if omitted, shall take the value of the domain from where this file was located.
DVBTriplet	The DVB triplet that can be used to refer to this service, even if the service is not delivered in a TS. Note: If this triplet matches the triplet of another service, it can be assumed that the services editorially carry the same content.
ServiceLocation	The location(s) where the A/V content for the service may be found. If multiple elements of this type are present, the one with the highest value of the priority attribute has the highest priority.
LCN	The logical channel number of the service.
TargetRegions	The target regions for the service where the service is intended to be received.
ServiceName	The name of the service. Multiple elements of this type may be provided as long as they all have different lang attributes.
ApplicationLocation	The location of the XML AIT file where the application associated with the service may be found, as defined in TS 102 809 [7].
ServiceGenre	The genre of the service.
ServiceType	The service type, as defined in EN 300 468 [10].
ContentAttributes	The attributes of the content for the service.
BCG	The details of a broadband content guide carrying metadata for this service.
IPServiceListType	A type to list all the available services and the BCG covering these services.
IPService	The details of the services.
BCG	The details of a broadband content guide carrying metadata for one or more services included in this list.
Version	The version number of this parent element.

D.1.3.3 IPServiceList and ServiceLocation

<element name="IPServiceList" type="dvb:IPServiceListType"/>
<element name="ServiceLocation" type="dvb:ServiceLocationType"/>

Element/Attribute	Semantics
IPServiceList	The list of services.
ServiceLocation	A top-level element containing a service location.

D.1.4 Example OSDT file (informative)

This example OSDT file contains one service with a logical channel number of 1 which can be provided either as SD or HD, both versions using MPEG DASH. A BCG service is also referenced, where EPG information for the service can be obtained.

```
<?xml version="1.0" encoding="UTF-8"?>
<dvb:IPServiceList Version="1" xmlns:dvb="urn:dvb:metadata:ciplus:osdt:2013" xmlns:tva="urn:tva:metadata:2005"</p>
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:dvb:metadata:ciplus:osdt:2013 osdt.xsd">
          <dvb:IPService>
                     <a href="cdvb:UniqueIdentifier"><a href="cdvb:UniqueIdentifier
                     <dvb:ServiceLocation>
                               <dvb:ContentAttributes>
                                         <dvb:AudioAttributes>
                                                    <tva:Coding href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:3.2">
                                                              <tva:Name>MPEG-1 Audio Layer II</tva:Name>
                                                   <tva:NumOfChannels>2</tva:NumOfChannels>
                                         </dvb:AudioAttributes>
                                         <dvb:VideoAttributes>
                                                   <tva:Coding href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:2.2.2">
                                                              <tva:Name>MPEG-2 Video Main Profile @ Main Level</tva:Name>
                                                    </tva:Coding>
                                         </dvb:VideoAttributes>
                               </dvb:ContentAttributes>
                               <a href="cdvb:UriBasedLocation">
<a href="dvb:UriBasedLocation">
<a href="cdvb:UriBasedLocation">
<
                                         <dvb:URI>http://www.broadcaster.com/mpd/channel-1-sd.mpd</dvb:URI>
                               </dvb:UriBasedLocation>
                    </dvb:ServiceLocation>
                     <dvb:ServiceLocation>
                               <dvb:ContentAttributes>
                                         <dvb:AudioAttributes>
                                                   <tva:Coding href="urn:dvb:metadata:cs:AudioCodecCS:2007:3.1">
                                                              <tva:Name>E-AC3</tva:Name>
                                                   </tva:Coding>
                                                   <tva:NumOfChannels>5</tva:NumOfChannels>
                                         </dvb:AudioAttributes>
                                         <dvb:VideoAttributes>
                                                   <tva:Coding href="urn:dvb:metadata:cs:VideoCodecCS:2007:1.4.12">
                                                              <tva:Name>H264 High Profile @ Level 4.0</tva:Name>
                                                    </tva:Coding>
                                         </dvb:VideoAttributes>
                               </dvb:ContentAttributes>
                               <a href="cdvb:UriBasedLocation">dvb:UriBasedLocation</a> contentType="application/dash+xml">
                                         <dvb:URI>http://www.broadcaster.com/mpd/channel-1-hd.mpd</dvb:URI>
                               </dvb:UriBasedLocation>
                    </dvb:ServiceLocation>
                    <dvb:LCN LCN="1" selectable="true"/>
                    <dvb:TargetRegions>
                              <a href="dvb:RegionList">dvb:RegionList</a> CountryCodes="GBR">
                                         <dvb:PrimaryRegion>
                                                   <dvb:Region>London</dvb:Region>
                                         </dvb:PrimaryRegion>
                               </dvb:RegionList>
                     </dvb:TargetRegions>
                     <dvb:ServiceName Language="eng">Channel 1</dvb:ServiceName>
          </dvb:IPService>
          <a href="www.operator.com">
                     <sds:BCG Id="BCG1" xmlns:sds="urn:dvb:metadata:iptv:sdns:2008-1">
                               <sds:Name Language="ENG">Operator BCG</sds:Name>
                               <sds:TransportMode>
                                         <sds:HTTP Location="http://www.operator/com/BCG1/dvb/sdns"/>
                               </sds:TransportMode>
                     </sds:BCG>
           </dvh·BCG>
 </dvb:IPServiceList>
```

Annex E (informative): Management of a DiSEqCTM switch under CICAM control

The aim of this annex is to facilitate improved user experience when operating on a satellite network, by offering an automatic installation procedure under the control of an operator-specific CICAM for the most common DiSEqCTM installations. This annex is purely informative.

When a CICAM sends a tune request to a host on a satellite network, the target satellite is specified in the satellite delivery descriptor using the orbital position. The tune command will succeed if a correctly configured LNB pointing to the expected satellite is connected to the receiver. However, if the LNB is connected to the Host using a DiSEqCTM switch, then the user has to set up the DiSEqCTM switch topology in the Host for the tune to succeed. This annex describes a simple discovery mechanism for DiSEqCTM switch topology. Here it is assumed that the LNB is of universal type and that the DiSEqCTM topology is a single DiSEqCTM switch with up to four positions. This will cover the most common cases and ensure a totally automated setup, even if the user is unaware of the antenna switch topology.

If a Host supports DiSEq C^{TM} , it will likely have a configuration menu with a table associating DiSEq C^{TM} positions with orbital positions (or at least satellite names). The discovery mechanism populates this table - the "Orbital-to-DiSEq C^{TM} " map - with correct information, when the CICAM issues a tune request. The assumption is that the user ensures that a given operator's transponders are accessible before inserting a CICAM from that operator. Since the parameters of the transponders are known, this approach will be much more efficient than a blind scan of all DiSEq C^{TM} positions.

Figure E.1 shows the discovery procedure.

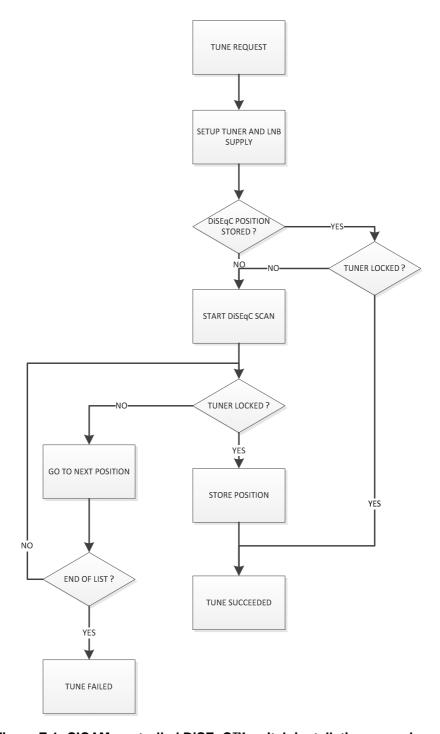


Figure E.1: CICAM-controlled DiSEqC™ switch installation procedure

- 1) The Host receives a tune command from the CICAM for one of the following reasons:
 - Broadcast tune in Host Control resource.
 - Tune request in Operator Profile resource.
 - User switching to a service defined in the CICAM NIT received in Operator Profile resource.
- 2) The Host sets tuner to the requested frequency and adjusts the 22 kHz tone and LNB voltage according to the requested frequency and polarization (supposing a universal LNB is connected to the antenna input).
- 3) If a satellite matching the requested orbital position is found in the "Orbital-to-DiSEq C^{TM} " map then the stored DiSEq C^{TM} command is sent to the antenna.

- 4) If lock is acquired then tune command is successful.
- 5) If lock is not acquired, the Host starts a scan of all DiSEqCTM positions, keeping the tuner on the same frequency. The positions to scan are: none, tone-burst A, tone-burst B, A, B, C, and D.
- 6) When a lock is acquired, the DiSEqCTM position is stored and the tune command is successful.
- 7) If no lock is acquired then the tune command fails.

Reliability can be increased by checking for delivery descriptors in the NIT-actual of found TSs.

If the Host has several satellite tuners, then a separate "Orbital-to-DiSEqCTM" map has to be compiled for each tuner. When attempting to tune, the Host may repeat the discovery procedure across all tuners, since the antenna switch topology may be different for each tuner.

Annex F (normative): CICAM application signalling

F.1 General

The contents of the present clause (annex F) will likely be incorporated into a future revision of the DVB application signalling specification [7]. When that revision has been published, the present clause (annex F) shall be considered to be deprecated. A future revision of the present document may have annex F removed and refer directly to the revised DVB application signalling specification [7] for the specification of methods relating to CICAM application signalling.

F.2 Signalling CICAM broadcast applications

When a CICAM broadcast application is to be signalled in the MPEG-2 table and section encoding of the AIT (see clause 5.3 of TS 102 809 [7]), this shall be done as defined in that specification with the following modifications:

- The application shall be signalled with a transport protocol descriptor where the *protocol_id* is 0x0004 and the selector bytes shall not be used.
- The *application_type* field shall indicate the middleware technology used by the application as registered with DVB. The application domain "CIEngineProfile1" defined in clause 12.2 of CI PlusTM 1.3 [3] and modified in the present document shall use the application type 0x0013.
- The application location descriptor shall reference an initial object provided by the CICAM. When the initial
 object is provided by the CICAM auxiliary file system, the domain identifier shall not be included, but shall be
 derived from the application type field.

It is recommended that the path to the initial object include the <code>organisation_id</code> and the <code>application_id</code>. This will avoid the Host accidentally starting the wrong application just because another application or CICAM happens to have an initial object of the same name in the same location. For example an application whose <code>organisation_id</code> was 0x21and whose <code>application_id</code> was 0x42 and whose initial object was called "index.html", might include the following path in the application location descriptor - "/0x21/0x42/index.html"

F.3 Relative priority of broadcast and CICAM broadcast applications

Where both broadcast and CICAM broadcast applications are signalled in the AIT for a service, the *application_priority* field of the AIT shall be used to decide which shall be started.

History

Document history				
V1.1.1	March 2014	Publication		