ETSI TS 103 205 V1.4.1 (2019-05)

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

EBU DVB®

Reference
RTS/JTC-DVB-383

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

Contents

Intellectual Property Rights 10
Foreword 10
Modal verbs terminology 10
Introduction 11
1 Scope 12
2 References 12
2.1 Normative references 12
2.2 Informative references 13
3 Definition of terms, symbols and abbreviations 14
3.1 Terms 14
3.2 Symbols 15
3.3 Abbreviations 15
4 CI Plus extensions overview 17
4.1 Introduction 17
4.2 Multi-stream reception 19
4.3 IP-delivered content 19
4.3.1 General 19
4.3.2 IP delivery modes 20
4.3.2.1 Host player mode 20
4.3.2.2 CICAM player mode 20
4.3.3 IP delivery use cases 21
4.3.3.1 General 21
4.3.3.2 Linear and VoD Streaming (pull) 21
4.3.3.3 Linear and VoD Streaming (push) 21
4.3.3.4 Downloaded content 21
4.4 CI Plus browser extensions 21
4.5 CICAM application launching 22
4.6 CICAM file retrieval 22
4.7 Usage Rules Information extensions 22
4.8 Watermarking and transcoding 22
5 General requirements 23
5.1 Backwards compatibility 23
5.2 Watermarking and transcoding 23
5.3 PES level scrambling 23
6 Multi-stream reception 24
6.1 General 24
6.2 TS Interface and Local TS multiplexing 25
6.2.1 Local TS identifier 25
6.2.2 Multiplexing broadcast and IP-delivered content 26
6.2.3 Multiplexed TS packet order, delay and delay variation 26
6.2.4 Scrambling cipher and CCK usage 26
6.2.5 Host Service Shunning 27
6.2.6 TS clock 27
6.2.7 Multi-stream operation with multiple CICAMs 27
6.3 PID Selection 27
6.3.1 General 27
6.3.2 Default PID selection 28
6.3.3 Default PID selection for frequency tune 28
6.3.4 PID selection priority 28
6.3.5 CICAM initiated update 28
6.3.6 Change in ES selection 29
6.3.7 Host initiated PID addition or removal 29

6.3.8 Selected services from the same broadcast TS 29
6.3.9 Example PID selection sequence 29
6.4 Resources for multi-stream operation 31
6.4.1 General 31
6.4.2 Multi-stream resource 31
6.4.2.1 General 31
6.4.2.2 CICAM multi-stream capability APDU 31
6.4.2.3 PID select request APDU 32
6.4.2.4 PID select reply APDU 33
6.4.3 Content Control resource 34
6.4.3.1 General 34
6.4.3.2 Content Control APDU extensions 34
6.4.3.2.1 cc_PIN_reply APDU 34
6.4.3.2.2 cc_PIN_event APDU 35
6.4.3.3 Content Control protocol extensions 35
6.4.3.3.1 URI transmission and acknowledgement protocol 35
6.4.3.3.2 Record Start protocol 36
6.4.3.3.3 Record Stop protocol extension 36
6.4.3.3.4 Change Operating Mode protocol extensions 37
6.4.3.3.5 CICAM to Host License Exchange protocol extensions 37
6.4.4 Conditional Access Support resource 38
6.4.4.1 General 38
6.4.4.2 ca_pmt APDU 38
6.4.4.3 ca_pmt_reply APDU 40
6.4.5 Multi-stream Host Control resource 40
6.4.5.1 General 40
6.4.5.2 tune_broadcast_req APDU 41
6.4.5.3 tune_triplet_req APDU 42
6.4.5.4 tune_lcn_req APDU 43
6.4.5.5 tune_ip_req APDU 43
6.4.5.6 tune_reply APDU 43
6.4.6 Application MMI resource 44
6.4.6.1 General 44
6.4.6.2 RequestStart APDU 44
6.4.6.3 RequestStartAck APDU 45
6.4.6.4 FileRequest APDU 45
6.4.6.5 FileAcknowledgeAPDU 45
6.4.6.6 AppAbortRequest APDU 45
6.4.6.7 AppAbortAck APDU 45
6.4.7 High-Level MMI resource 45
6.4.7.1 General 45
6.4.7.2 enq APDU 45
6.4.7.3 answ APDU 46
6.4.7.4 menu APDU 47
6.4.7.5 menu_answ APDU 47
6.4.7.6 list APDU 48
6.4.7.7 close_mmi APDU 48
6.4.7.8 display_control APDU 48
6.4.7.9 display_reply APDU 48

7 IP delivery Host player mode 49
7.1 General 49
7.2 TS interface modes 49
7.3 Command interface 49
7.3.1 General 49
7.3.2 Playback initiation 50
7.3.3 Playback execution 51
7.3.4 Playback termination 52
7.4 Sample decryption resource 52
7.4.1 Resource usage 52
7.4.2 sd_info_req APDU 53
7.4.3 sd_info_reply APDU 53

7.4.4 sd_start APDU 54
7.4.5 sd_start_reply APDU 57
7.4.6 sd_update APDU 58
7.4.7 sd_update_reply APDU 60
7.5 TS interface 60
7.5.1 General 60
7.5.2 TS content carriage 60
7.5.2.1 Host output 60
7.5.2.2 CICAM output 62
7.5.2.3 Multiple TS Sample Tracks 63
7.5.3 Non-TS content carriage 63
7.5.3.1 General 63
7.5.3.2 TS packet padding 64
7.5.3.3 TS packet PAD signalling 65
7.5.3.4 Host output 66
7.5.3.4.1 General 66
7.5.3.4.2 Transmitting Samples 66
7.5.3.4.3 Managing multiple Tracks 68
7.5.3.4.4 Track list update 70
7.5.3.5 TS packets 72
7.5.3.6 CICAM output 73
7.5.3.7 ISOBMFF Samples 74
7.5.3.7.1 Sample Start TS Packet (SSP) 74
7.5.3.7.2 ISOBMFF Sample packetization 74
7.5.4 CICAM buffering 75
7.5.4.1 General 75
7.5.4.2 Buffer size indication 75
7.5.4.3 Buffer level management 75
7.5.4.4 Flushing CICAM buffers 75
7.5.5 Messages and descriptors 75
7.5.5.1 Introduction 75
7.5.5.2 General messages 76
7.5.5.2.1 General 76
7.5.5.2.2 Coding of table_id values 76
7.5.5.2.3 Flush Table (FLT) 76
7.5.5.3 Track-related messages 77
7.5.5.3.1 General 77
7.5.5.3.2 Sample Start TS Packet (SSP) 78
7.5.5.3.3 Sample End TS Packet (SEP) 78
7.5.5.4 Descriptors 79
7.5.5.4.1 General 79
7.5.5.4.2 CI Plus initialization vector descriptor 79
7.5.5.4.3 CI Plus key identifier descriptor 80
7.6 URI 80

8 IP delivery CICAM player mode 80
8.1 General 80
8.2 Player controls 81
8.3 Session initialization 81
8.3.1 General 81
8.3.2 Host-initiated playback 81
8.3.3 CICAM-initiated playback 82
8.4 Communication errors 82
8.5 Trick mode support 82
8.6 Session termination 83
8.6.1 General 83
8.6.2 Unrecoverable error 83
8.6.3 Termination by the user 83
8.6.4 End of content 83
8.7 CICAM player mode sequence 83
8.8 CICAM Player resource 86
8.8.1 General 86

8.8.2 CICAM Player resource APDUs 86
8.8.3 CICAM_player_verify_req APDU 86
8.8.4 CICAM_player_verify_reply APDU 87
8.8.5 CICAM_player_capabilities_req APDU 87
8.8.6 CICAM_player_capabilities_reply APDU 88
8.8.7 CICAM_player_start_req APDU 88
8.8.8 CICAM_player_start_reply APDU 89
8.8.9 CICAM_player_play_req APDU 90
8.8.10 CICAM_player_status_error APDU 91
8.8.11 CICAM_player_control_req APDU 91
8.8.12 CICAM_player_info_req APDU 93
8.8.13 CICAM_player_info_reply APDU 93
8.8.14 CICAM_player_stop APDU 93
8.8.15 CICAM_player_end APDU 94
8.8.16 CICAM_player_asset_end APDU 94
8.8.17 CICAM_player_update_req APDU 95
8.8.18 CICAM_player_update_reply APDU 95
8.8.19 CICAM player resource summary 96

9 CICAM file retrieval 96
9.1 General 96
9.2 File system offer APDU 97
9.3 File System Ack APDU 98
9.4 File request APDU 99
9.5 File acknowledge APDU 99
9.6 Auxiliary file system resource summary 99
9.7 Auxiliary file system and Application MMI resource coordination 99

10 Low Speed Communication resource version 4 99
10.1 General 99
10.2 Host IP configuration information 100
10.3 Information about IP streams 100
10.4 IP multicast 101
10.5 Source ports 101
10.6 Host to CICAM delivery 101
10.7 CICAM to Host delivery 101
10.8 IP flow control 101
10.9 comms_info 102
10.9.1 General 102
10.9.2 comms_info_req APDU 102
10.9.3 comms_info_reply APDU 102
10.10 comms_IP_config 103
10.10.1 General 103
10.10.2 comms_IP_config_req 103
10.10.3 comms_IP_config_reply 103
10.11 Comms Cmd modification 105
10.11.1 General 105
10.11.2 Comms Cmd hybrid_descriptor 106
10.11.3 Comms Cmd multicast_descriptor 107
10.12 Low Speed Communications resource types modification 107
10.12.1 General 107
10.12.2 CICAM Flow Control 108
10.12.3 Disconnection behaviour 108
10.12.4 Data transfer across the TS interface 108
10.12.4.1 TS packet syntax 108
10.12.4.2 Adaptation field usage 109
10.12.4.3 Detection of last fragment of last UDP Packet on CICAM side (informative) 110

11 Usage Rules Information version 3 110

12 CICAM applications 112
12.1 General 112
12.2 CI Plus Browser extensions 112

12.2.1 InteractionChannelExtension 112
12.2.2 ICStreamingExtension 112
12.2.3 ICEncryptedStreamExtension 112
12.2.4 Video scaling 112
12.2.4.1 General 112
12.2.4.2 Set of features 113
12.2.4.3 GetEngineSupport 113
12.3 Application life cycle 113
12.4 Application Coordination Framework 115
12.4.1 General 115
12.4.2 Modifications to referenced specifications 115
12.4.3 CICAM application launch mechanisms 116
12.4.3.1 General 116
12.4.3.2 CICAM AppMMI applications 116
12.4.3.3 CICAM broadcast applications 117
12.4.3.3.1 General 117
12.4.3.3.2 Signalling CICAM broadcast applications in the broadcast stream 117
12.4.3.3.3 Advertising CICAM broadcast applications 117
12.4.4 Broadcast and CICAM application coordination 118
12.4.4.1 General 118
12.4.4.2 Changing to a DVB service 119
12.4.4.3 An application is running and has focus 119
12.4.4.4 Changing to the CICAM Virtual Channel 120
12.4.4.5 enter_menu 120
12.4.4.6 Application termination 120
12.4.5 Application coordination scenarios 120
12.4.5.1 General 120
12.4.5.2 Broadcast signalled application launch 120
12.4.5.3 CICAM AppMMI application launch 122
12.4.5.4 Broadcast application launches CICAM broadcast application 124
12.5 Host application environments 125
12.5.1 General 125
12.5.2 Application provision on a CICAM 125
12.5.3 Determining Host application environment support 126

13 DVB Host Control version 3 126
13.1 General 126
13.2 DVB Host Control version 3 APDUs 126
13.2.1 tune_broadcast_req APDU 126
13.2.2 tune_triplet_req APDU 127
13.2.3 tune_lcn_req APDU 128
13.2.4 tune_ip_req APDU 129
13.2.5 tuner_status_req APDU 129
13.2.6 tuner_status_reply APDU 129

14 CICAM Virtual Channel 130
14.1 Introduction 130
14.2 Application Information version 4 131
14.2.1 General 131
14.2.2 enter_cicam_channel APDU 131

15 Operator Profile version 2 131
15.1 Introduction 131
15.2 Profile Type 2 132
15.2.1 Introduction 132
15.2.2 Initialization and profile discovery 132
15.2.3 Moving between profiles 132
15.2.4 LCN collision management 133
15.3 CICAM Virtual Channel 133
15.3.1 CICAM Virtual Channel descriptor 133
15.3.2 Selection of the CICAM Virtual Channel 134
15.3.3 Operator NIT management APDU 134
15.4 IP delivered services 136

15.4.1 operator_status() APDU 136
15.4.2 IP service discovery and update 136
15.4.3 operator_osdt_request APDU 137
15.4.4 operator_osdt_reply APDU 137

16 High-Level MMI 137

17 Operator Profile version 3 138
17.1 General 138
17.2 Operator Profile version 3 APDUs 138
17.2.1 operator_codec_verify_req APDU 138
17.2.2 operator_codec_verify_reply APDU 139
17.3 ciplus_advanced_service_descriptor 139

18 CICAM Player version 2 140
18.1 General 140
18.2 CICAM Player version 3 APDUs 140
18.2.1 CICAM_player_codec_verify_req APDU 140
18.2.2 CICAM_player_codec_verify_reply APDU 141

19 Usage Rules Information (URI) version 5 142

20 Clarifications on usage of Content Control System ID 144

21 Application Information version 5 144
21.1 General 144
21.2 APDU extensions 145
21.2.0 Application Information resource summary 145
21.2.1 Host diagnostic screen APDUs 145
21.2.1.0 General 145
21.2.1.1 HDS request APDU 145
21.2.1.2 HDS confirmation APDU 146
21.2.2 Power down APDUs 146
21.2.2.0 General 146
21.2.2.1 Power down notice APDU 146
21.2.2.2 Power down OK APDU 147

22 Content Control version 4 (multi-stream version 2) 147
22.1 General 147
22.2 Protocol extensions 148
22.2.1 Usage Rules Information (URI) version 4 148
22.2.2 Output Control protocol 149

23 Content Control version 5 (multi-stream version 3) 150
23.1 Negotiation of optional features 150
23.1.1 General 150
23.1.2 Negotiation protocol for optional features 150
23.1.3 Data type for optional features 151

24 Optional Content Control Features 151
24.1 List of optional features 151
24.2 Overt Watermarking 151
24.2.1 General 151
24.2.2 Overt watermark content 151
24.2.3 Logical plane, rendering and displaying 152
24.2.4 Overt watermarking data types 153
24.2.4.1 wm_instructions (CICAM → Host) 153
24.2.4.2 wm_rendered (Host → CICAM) 153
24.2.4.3 wm_off (CICAM → Host) 154
24.2.5 Overt watermarking protocol 154
24.2.5.1 Displaying an overt watermark 154
24.2.5.2 Host status updates 155
24.2.5.3 CICAM removing an overt watermark 155
24.2.5.4 Host removing an overt watermark 156
24.2.6 Overt watermarking during recording and playback 156

24.2.7 Overt watermarking during playback of IP-delivered content 156
24.2.7.1 General 156
24.2.7.2 Overt watermarking in CICAM player mode 156
24.2.7.3 Overt watermarking in Host player mode 157
24.2.8 Overt watermarking sequence (informative) 157

Annex A (normative): Resources summary 158

Annex B (informative): Credential Specification: Parameters exchanged in APDUs 164

Annex C (informative): Descriptor identification and location 166

Annex D (normative): Schemas 167

D.1 OSDT schema 167
D.1.1 General 167
D.1.2 Namespace 167
D.1.3 Complex types and attribute groups 167
D.1.3.1 SubRegionType 167
D.1.3.2 ServiceLocationType 168
D.1.3.3 IPServiceList and ServiceLocation 170
D.1.4 Example OSDT file (informative) 170

Annex E (informative): Management of a DiSEqC™ switch under CICAM control 172

Annex F (normative): CICAM application signalling 175

F.1 General 175

F.2 Signalling CICAM broadcast applications 175

F.3 Relative priority of broadcast and CICAM broadcast applications 175

Annex G (informative): Bibliography 176

Annex H (informative): Change History 177

History 178

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables 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 (https://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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

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.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The DVB Common Interface specifications CENELEC EN 50221 [1] and ETSI 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 Plus specification [3], developed by CI Plus 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 Plus V1.3 specification [3], which was produced and continues to be published by CI Plus LLP.

2 References

2.1 Normative 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.

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] ETSI TS 101 699 (V1.1.1) (11-1999): "Digital Video Broadcasting (DVB); Extensions to the Common Interface Specification".

[3] CI Plus 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 (2014)/ISO/IEC 13818-1:2015: "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) (03-2013): "MHEG-5 Broadcast Profile".

[7] ETSI TS 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: "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, September 2012.

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

[13] Open IPTV Forum: "Release 1 Specification, Volume 3 - Content Metadata", V1.2, September 2012.

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

[14] ISO/IEC 23001-7:2016: "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: https://www.digital-cp.com/hdcp-specifications.

[25] "Digital Transmission Content Protection Specification Volume 1 (Informational Version)", Revision 1.71.

NOTE: Available from: http://www.dtcp.com/specifications.aspx.

[26] Void.

[27] "High-bandwidth Digital Content Protection System", Revision 1.4.

NOTE: Available from: https://www.digital-cp.com/hdcp-specifications.

[28] IETF RFC 4291: "IP Version 6 Addressing Architecture".

2.2 Informative 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.

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

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".

[i.2] A173-2: "Second Generation Common Interface (CI); Part 2: Extensions to the CI Plus Specification (CI Plus 2.0)", June 2015.

NOTE: Available from https://www.dvb.org/resources/public/standards/a173-2_ci_plus_2_-_part_2.pdf.

[i.3] CI Plus Specification v1.4.3 (11-2017): "CI Plus Specification. Content Security Extensions to the Common Interface. v1.4.3".

NOTE: Available from http://www.ci-plus.com/data/ci-plus_specification_v1.4.3.pdf.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

application coordination framework: set of CI Plus 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 Plus 1.3 [3].

NOTE: CI Plus V1.3 [3] also uses this term to mean the CI Plus browser. ETSI 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 AppMMI application: application launched by the CICAM using the CI Plus Application MMI resource

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

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 Plus 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 Symbols

Void.

3.3 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
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 Initialization 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
DMA	Direct Memory Access
DNS	Domain Name Service
DOT	Digital Only Token

DRM	Digital Rights Management
DSD	Delivery System Descriptor
DTCP	Digital Transmission Content Protection
DTV	Digital Television
DVB	Digital Video Broadcasting
DVB-C	DVB Cable
DVB-S	DVB Satellite
DVB-T	DVB Terrestrial
ECM	Entitlement Control Message
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
IETF	Internet Engineering Task Force

NOTE: See https://ietf.org/.

IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPTV	IP Television
IRD	Integrated Receiver Decoder
ISO	International Standards Organization
ISOBMFF	ISO Base Media File Format
IV	Initialization 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
PAD	Padding
PAT	Program Association Table
PCMCIA	PC Memory Card International Association
PES	Packetized Elementary Stream
PID	Packet IDentifier
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PMT	Program Map Table
PSI	Program Specific Information

PVR	Personal Video Recorder
RCT	Redistribution Control Token
RFC	Request for Comments
ROT	Root of Trust (i.e. Trust Authority)
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol
SAC	Secure Authenticated Channel
SAS	Specific Application Support
SD	Standard Definition
SDT	Service Description Table
SEP	Sample End TS Packet
SI	Service Information
SPTS	Single Programme Transport Stream
SRM	System Renewability Message
SSM	Set Subtitle Mode
SSP	Sample Start TS Packet
TCP	Transmission Control Protocol
TS	MPEG-2 Transport Stream
TV	Television
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 Plus 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 Plus 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 Plus browser extensions.
- CICAM application launching.
- URI (usage rules information) extensions.
- Watermarking and transcoding capability.

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

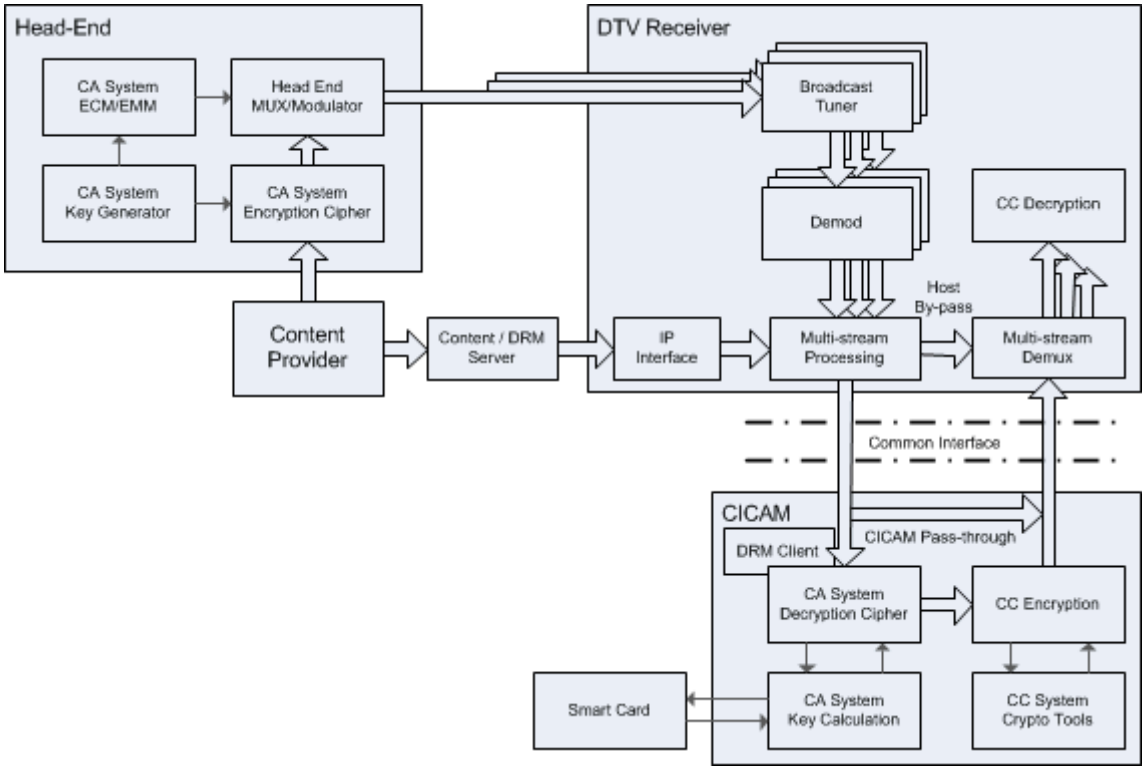

Figure 1: CI Plus system overview

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 Plus V1.3 [3], to include the new feature of IP delivery, where content is nominally protected by a DRM system.

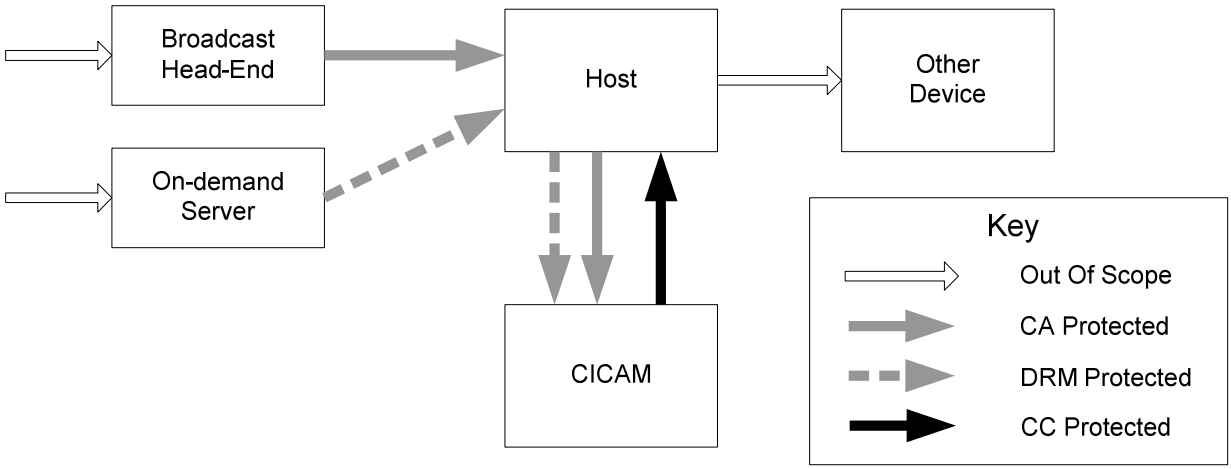

Figure 2: Scope of protection schemes

The present document does not contain any extensions to the following features specified in CI Plus 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 Plus man machine interface resource (specified in clause 13 of [3]).

The present document extends annex H of the CI Plus 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 Plus V1.3 specification [3] are amended in the present document. In particular, no extensions or changes are made to the CI Plus electrical specification, contained in annex K of the CI Plus 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 Plus that are introduced with the present document. These consist of completely new features as well as amendments and restrictions against CI Plus 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™ installations.

4.2 Multi-stream reception

The CI Plus 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 Plus 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 Plus 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 Plus Content Control system already defined in the CI Plus 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 Plus 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 UDP/TCP payload data 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 initialization 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 Plus browser extensions

The CI Plus browser is extended to include the functionality of static content retrieval over an IP connection by a CI Plus 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 12.2 of the present document.

In CI Plus V1.3 [3] it is optional for the Host to support scaled video within a CI Plus browser application. The present document specifies that scaled video shall be supported in the CI Plus browser. The corresponding affected amendments for this aspect are specified in clause 12.2.4.

Host devices increasingly include hybrid broadcast broadband capability. The present document contains extensions to CI Plus that enable the CI Plus Browser to make use of this capability. Extensions are also defined to enable improvements in the user experience of operator applications using the CI Plus 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 Plus 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 CENELEC 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 Plus, i.e. V1.3 [3] and earlier, and with CICAMs conforming to DVB CI [1] and [2].

Particular issues concerning multi-stream operation 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 Plus, 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 Plus 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 Plus 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 Plus architecture to facilitate multi-stream support, as an informative example of a Host device with three broadcast tuners and IP delivery capability.

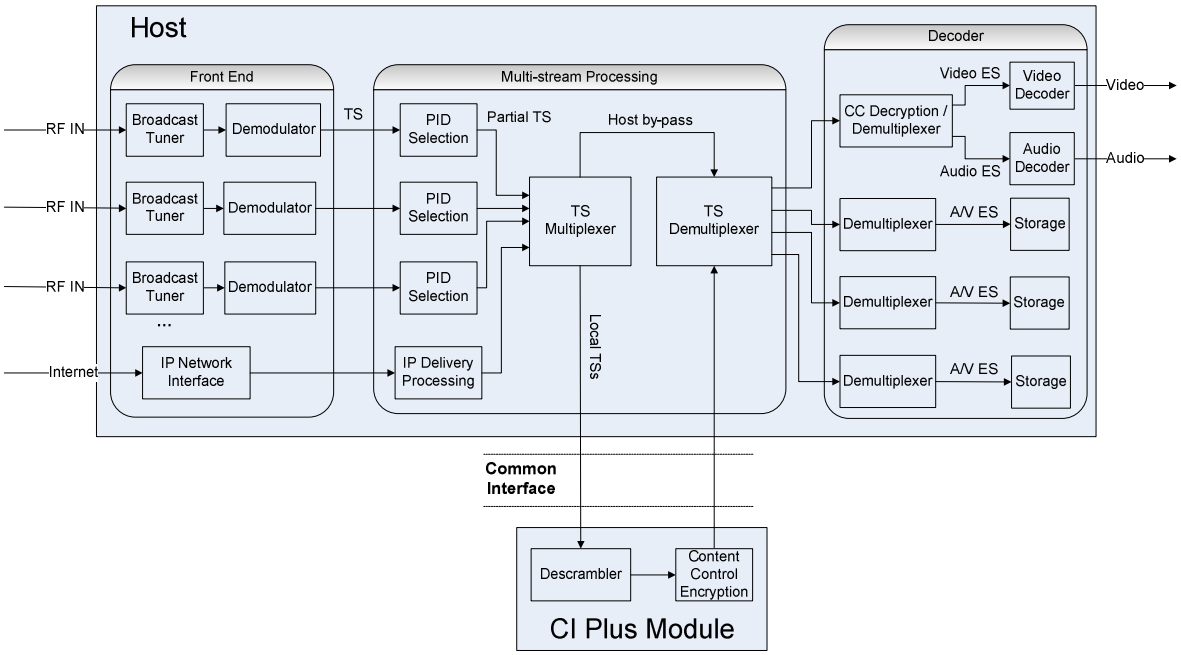

Figure 3: CI Plus extended architecture for multi-stream reception

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 Plus 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 initialization 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 Plus [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 Plus 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 standardized 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 Plus 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 (see annex A of CENELEC EN 50221 [1]) 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 does not need to 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 regarding 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.

Table 1: TS packet order, delay and delay variation

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

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 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 Initialization 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 Plus 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 Plus 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 ETSI EN 300 468 [10].
- The SDT PID, as defined in ETSI 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_req()* 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 prioritize 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.

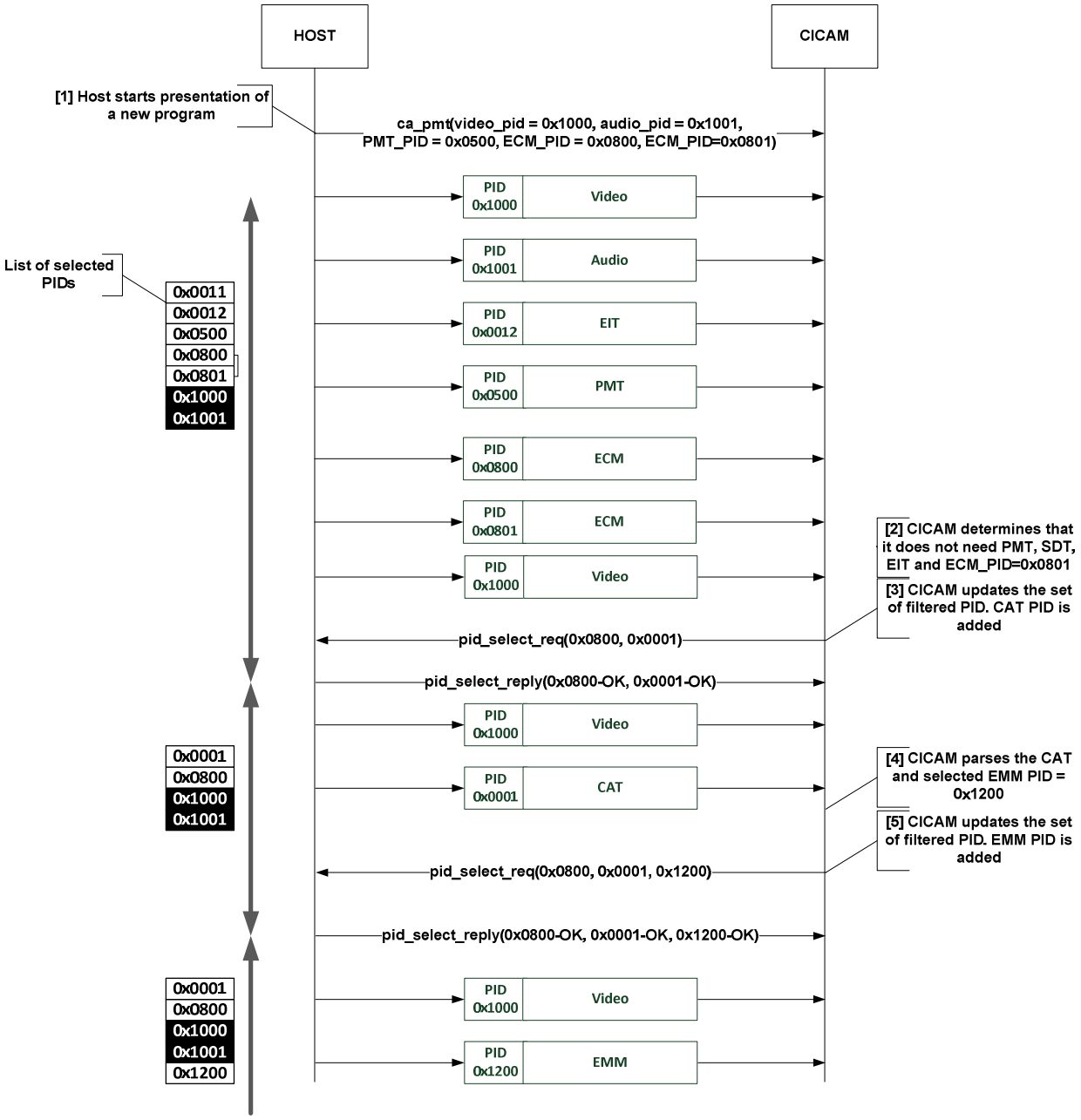

Figure 4: PID selection example sequence diagram

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_descriptor*s 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**	**Type**	**Ver.**	**APDU Tag**	**Tag value**	**Host**	**CICAM**
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	→	

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 CENELEC 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 descending priority order. Each PID has a *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 c*ritical_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 CENELEC 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_selected_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.

Table 5: PID_select_reply APDU

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

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 CENELEC 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

Resource					Application Object		Direction	
Name	**Resource Identifier**	**Class**	**Type**	**Ver.**	**APDU Tag**	**Tag value**	**Host**	**CICAM**
Content Control	00 8C 10 41	140	65	1	cc_open_req	9F 90 01	←	
					cc_open_cnf	9F 90 02	→	
					cc_data_req	9F 90 03	←	
					cc_data_cnf	9F 90 04	→	
					cc_sync_req	9F 90 05	←	
					cc_sync_cnf	9F 90 06	→	
					cc_sac_data_req (note 1)	9F 90 07	←→	
					cc_sac_data_cnf (note 1)	9F 90 08	←→	
					cc_sac_sync_req	9F 90 09	←	
					cc_sac_sync_cnf	9F 90 10	→	
					cc_PIN_capabilities_req	9F 90 11	→	
					cc_PIN_capabilities_reply	9F 90 12	←	
					cc_PIN_cmd	9F 90 13	→	
					cc_PIN_reply (note 2)	9F 90 14	←	
					cc_PIN_event (note 2)	9F 90 15	←	
					cc_PIN_playback	9F 90 16	→	
					cc_PIN_MMI_req	9F 90 17	→	
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 Plus 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_reply_tag	24	uimsbf
length_field()		
reserved	7	uimsbf
LTS_bound_flag	1	uimsbf
if (LTS_bound_flag == 1) {		
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 CENELEC 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_req()* APDUs.

LTS_id: Local TS identifier.

PINcode_status_field: Refer to clause 11.3.2.3 of the CI Plus 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.

Table 8: cc_PIN_event APDU syntax

Syntax	Number of bits	Mnemonic
cc_PIN_event() {		
cc_PIN_event_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
program_number	16	uimsbf
PINcode_status_field	8	uimsbf
rating	8	uimsbf
pin_event_time_utc	40	uimsbf
pin_event_time_centiseconds	8	uimsbf
private_data	8x15	uimsbf
}		

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 CENELEC EN 50221 [1], clause 8.3.1.
- **LTS_id:** Local TS identifier.
- **Other fields:** Refer to clause 11.3.2.4 of the CI Plus 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 *LTS_id* data type shall use a *datatype_id* value of 50.

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 to the Host	cc_sac_data_req	send_datatype_nbr = 3		
			I	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 acknowledgement to the CICAM	cc_sac_data_cnf	send_datatype_nbr = 1		
			I	datatype_id	datatype_len
			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 of start of recording	cc_sac_data_req	send_datatype_nbr = 3 or 4		
			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_nbr = 1		
			i	datatype_id	
			0	40 (record_start_status)	
2	CICAM sends acknowledgement to the Host	cc_sac_data_cnf	send_datatype_nbr = 1		
			i	datatype_id	datatype_len
			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 recording has stopped	cc_sac_data_req	send_datatype_nbr = 2		
			i	datatype_id	datatype_len
			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 an acknowledgement to the Host	cc_sac_data_cnf	send_datatype_nbr = 1		
			i	datatype_id	datatype_len
			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 of change of operating mode	cc_sac_data_req	send_datatype_nbr = 3		
			i	datatype_id	datatype_len
			0	38 (operating_mode)	8 bits
			1	26 (program_number)	16 bits
			2	50 (LTS_id)	8 bits
			request_datatype_nbr = 1		
			i	datatype_id	
			0	41 (mode_change_status)	
2	CICAM sends an acknowledgement to the Host	cc_sac_data_cnf	send_datatype_nbr = 1		
			i	datatype_id	datatype_len
			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

Step	Action	APDU	Content		
1	CICAM supplies the Host with content license	cc_sac_data_req	send_datatype_nbr = 4 or 5		
			i	datatype_id	datatype_len
			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)	
2	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
NOTE 1: The program_number matches the Record Start message's program_number. NOTE 2: Table 11.45 in the CI Plus V1.3 specification [3] contains the allowed values and meaning of this field. 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	Number 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 */		
}		
}		
}		
}		

The fields are defined as follows:

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	Number 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 13. 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 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 close the session 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	Direction	
	Host	CICAM
tune_broadcast_req	←	
tune_triplet_req	←	
tune_lcn_req	←	
tune_ip_req	←	
tune_reply	→	
ask_release	→	
ask_release_reply	←	
tuner_status_req	←	
tuner_status_reply	→	

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 a given session, a CICAM shall not request a further tune before the Host has replied to the previous request.

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 13. The following clauses specify the modified syntaxes of the relevant APDUs for multi-stream operation.

6.4.5.2 tune_broadcast_req 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
tune_broadcast_req(){		
tune_broadcast_req_tag	24	uimsbf
length_field()		
reserved	4	uimsbf
background_tune_flag	1	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
pmt_flag	1	uimsbf
service_id	16	uimsbf
reserved	4	uimsbf
descriptor_loop_length	12	uimsbf
for (i=0; i<N; i++){		
descriptor()		
}		
if (pmt_flag == 1) {		
program_map_section()		
}		
}		

The fields are defined as follows:

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 13.2.1.

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
tune_triplet_req () {		
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
if (delivery_system_descriptor_tag == 0x7f){		
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 13.2.2.

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
tune_lcn_req () {		
tune_lcn_req_tag	24	uimsbf
length_field()		
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 13.2.3.

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
tune_ip_req () {		
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++) {		
service_location_data	8	uimsbf
}		
}		

background_tune_flag: Refer to clause 6.4.5.2.

Other Fields: Refer to clause 13.2.4.

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 Plus 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.

Table 23: RequestStart APDU syntax

Syntax	Number of bits	Mnemonic
RequestStart(){		
RequestStart_tag	24	uimsbf
length_field()		
reserved	7	bslbf
LTS_bound_flag	1	bslbf
if (LTS_bound_flag == 1) {		
LTS_id	8	uimsbf
} else {		
reserved	8	bslbf
}		
AppDomainIdentifierLength	8	uimsbf
InitialObjectLength	8	uimsbf
for (i=0; i<AppDomainIdentifierLength; i++){		
AppDomainIdentifier	8	uimsbf
}		
for (i=0; i< InitialObjectLength; i++){		
InitialObject	8	uimsbf
}		
}		

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 ETSI 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 ETSI TS 101 699 [2], clause 6.5.4.

6.4.6.5 FileAcknowledgeAPDU

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

6.4.6.6 AppAbortRequest APDU

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

6.4.6.7 AppAbortAck APDU

The *AppAbortAck()* APDU is as defined in ETSI 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 enq 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(){		
enq_tag	24	uimsbf
length_field()		
reserved	6	bslbf
LTS_bound_flag	1	bslbf
blind_answer	1	bslbf
if (LTS_bound_flag == 1) {		
LTS_id	8	uimsbf
}else {		
reserved	8	bslbf
}		
answer_text_length	8	uimsbf
for (i=0; i<enq_length - 3; i++){		
text_char	8	uimsbf
}		
}		

The fields are defined as follows:

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 CENELEC EN 50221 [1].

6.4.7.3 answ APDU

The *answ()* APDU is as defined in CENELEC 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	bslbf
if (LTS_bound_flag == 1) {		
LTS_id	8	uimsbf
}else {		
reserved	8	bslbf
}		
choice_nb	8	uimsbf
TEXT() /* title text */		
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i<choice_nb; i++){ /* when choice_nb != 'FF' */		
TEXT()		
}		
}		

The fields are defined as follows:

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 CENELEC EN 50221 [1].

6.4.7.5 menu_answ APDU

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

Table 28: choice_ref 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" *choice_ref*, 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
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 {		
reserved	8	bslbf
}		
item_nb	8	uimsbf
TEXT() /* title text */		
TEXT() /* sub-title text */		
TEXT() /* bottom text */		
for (i=0; i<item_nb; i++){ /* when item_nb != 'FF' */		
TEXT()		
}		
}		

The fields are defined as follows:

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 CENELEC EN 50221 [1].

6.4.7.7 close_mmi APDU

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

6.4.7.8 display_control APDU

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

6.4.7.9 display_reply APDU

The *display_reply()* APDU is as defined in CENELEC 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 Plus 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 Plus 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 Plus 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 related 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 does not need to 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 does not need to 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 two-stage acknowledgement mechanism, depicted in Figure 5, provides an opportunity to accelerate the presentation of the content and improve the user experience.

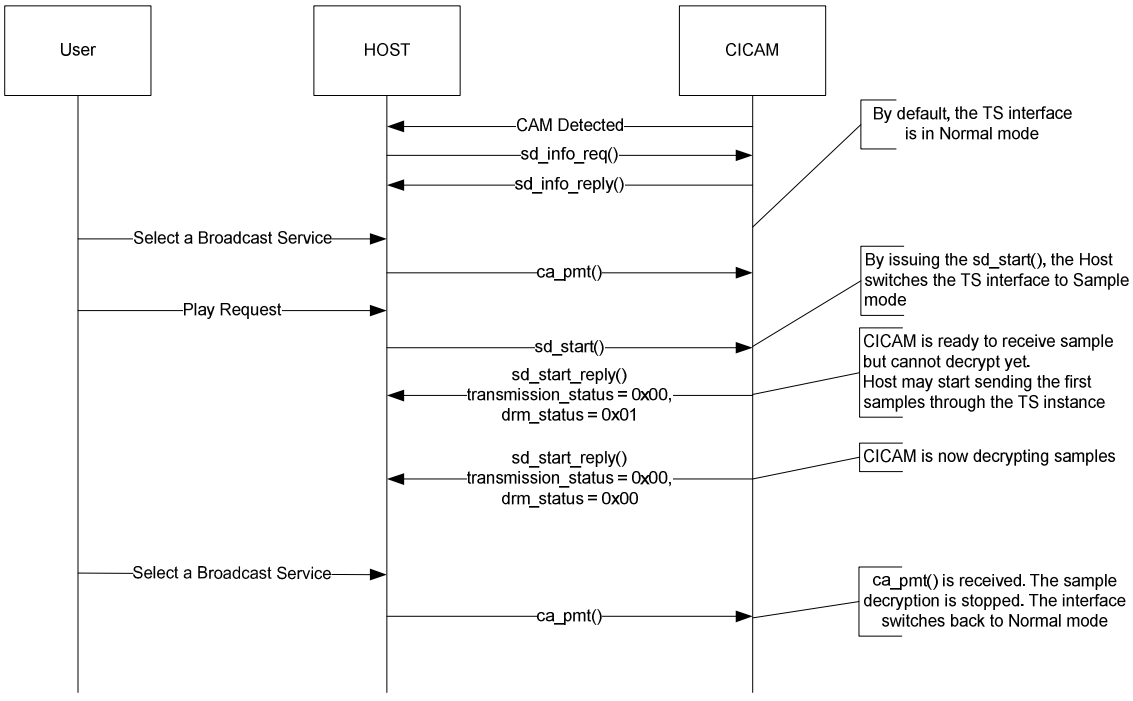

Figure 5: Two-stage playback initiation

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.

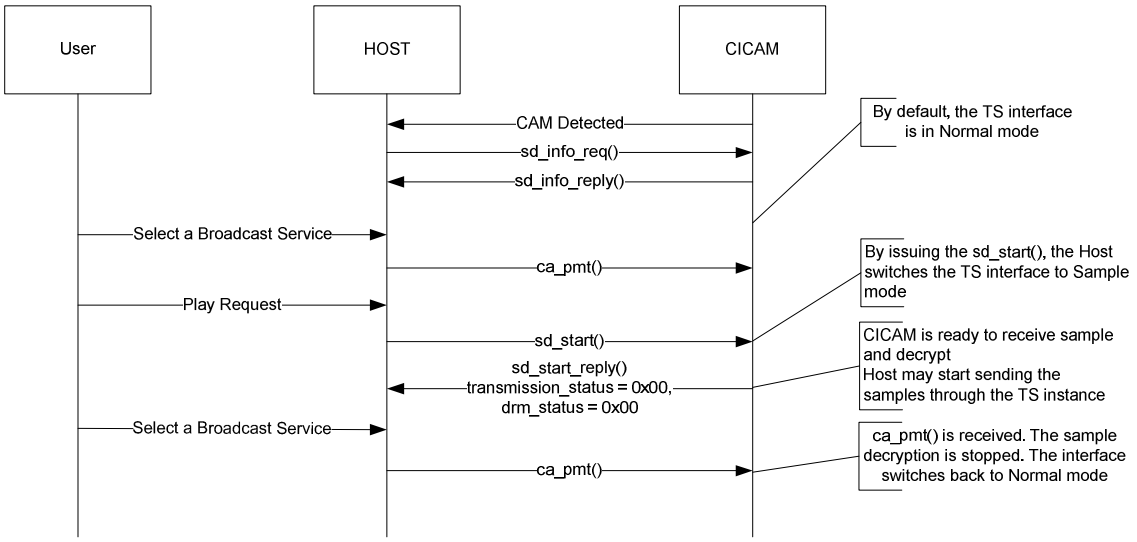

Figure 6: Single-stage playback initiation

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, e.g. the user may change the audio language, which would 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.

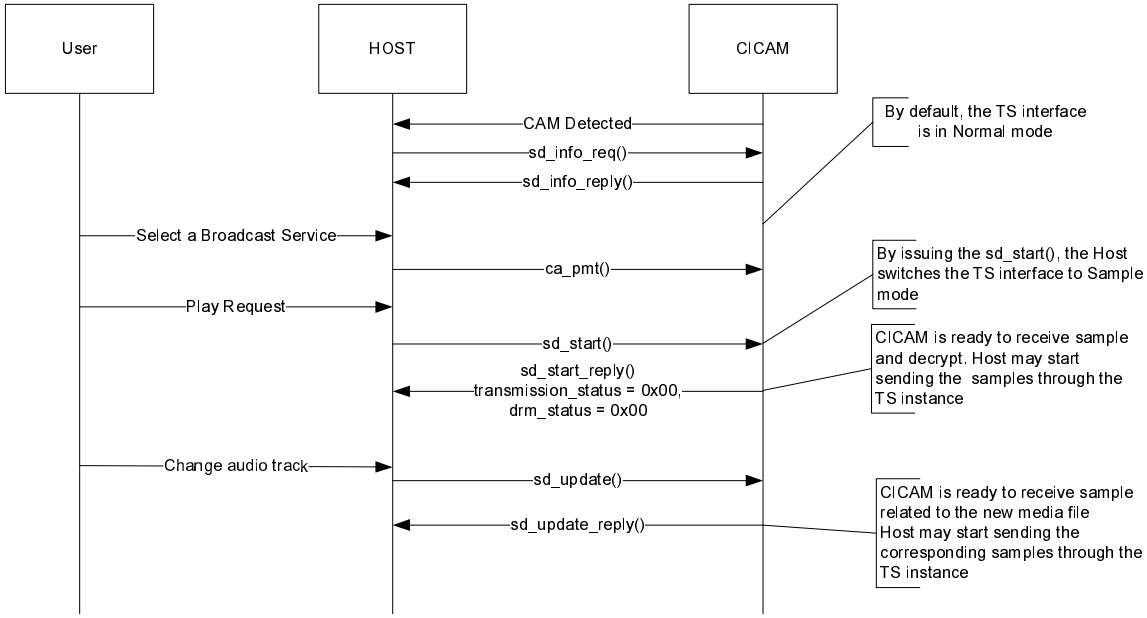

Figure 7: Playback execution sequence diagram

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.

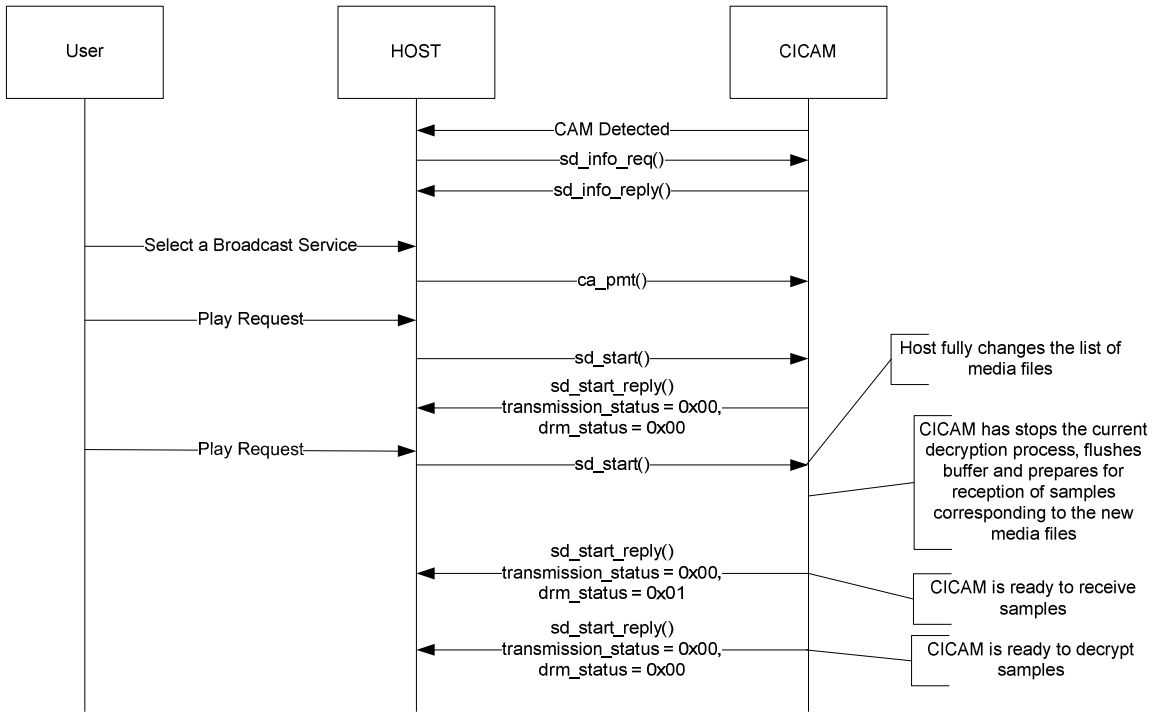

Figure 8: Playback re-start sequence diagram

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

Resource					Application Object		Direction	
Name	**Resource Identifier**	**Class**	**Type**	**Vers.**	**APDU Tag**	**Tag value**	**Host**	**CICAM**
Sample Decryption	00920041	146	1	1	sd_info_req	9F 98 00	→	
					sd_info_reply	9F 98 01	←	
					sd_start	9F 98 02	→	
					sd_start_reply	9F 98 03	←	
					sd_update	9F 98 04	→	
					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

sd_info_req() {
  sd_info_req_tag
  length_field()
}
	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 CENELEC 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

sd_info_reply() {
  sd_info_reply_tag
  length_field()
  number_of_drm_system_id
  for (i=0; i<n; i++) {
  drm_system_id
  }
  number_of_drm_uuid
  for (i=0; i<n; i++) {
  drm_uuid
  }
}
	24
	16
	8
	
	16
	
	8
	
	128
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf

The fields are defined as follows:

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 CENELEC 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 CENELEC EN 50221 [1] and CI Plus 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 until the next *sd_start()* APDU or *ca_pmt()* APDU. 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++) {		
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++) {		
drm_metadata_byte	8	uimsbf
}		
}		
}		
if (ts_flag == 0){		
number_of_sample_tracks	8	uimsbf
for (i=0;i<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++) {		
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++) {		
drm_metadata_byte	8	uimsbf
}		
}		
}		
}		
}		

The fields are defined as follows:

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 CENELEC 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. 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 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].	0x01
Content Access Streaming Descriptor (CASD) - DRM Private Data. The DRMPrivateData 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 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].	0x02
Common Encryption (CENC) - Protection System Specific Header ('pssh') box. 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].	0x03
Media Presentation Description (MPD) - Content Protection Element. The ContentProtection element from the Representation element for the Track to be decrypted is copied into the drm_metadata_byte array as a UTF-8 encoded 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.	0x04
ISOBMFF - Protection Scheme Information ('sinf') box. 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].	0x05
Online SDT (OSDT) - DRM Generic Data. 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].	0x06
Online SDT (OSDT) - DRM Private Data. 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].	0x07
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.
- Any transfer block size requirement of the CICAM to enable the host to ensure that all data has been moved through the CICAM pipeline.

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()*.

Sometimes additional TS packets may be required to force a DMA transfer within the CICAM, thereby forcing the processing of any residue data that exists in the CICAMs internal buffering system. Hence, when a CICAM reports a data_block_size greater than zero, a Host may deliver additional null packets when it wishes the CICAM to process residual Sample data, which is smaller than the indicated data_block_size.

Table 35: sd_start_reply APDU syntax

Syntax	Number of bits	Mnemonic
sd_start_reply() {		
sd_start_reply_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
transmission_status	8	uimsbf
drm_status	8	uimsbf
drm_system_id	16	uimsbf
drm_uuid	16*8	uimsbf
buffer_size	16	uimsbf
data_block_size	16	uimsbf
}		

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 CENELEC 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.

data_block_size: This 16-bit field indicates the number of transport stream packets that should be sent to the CICAM to ensure that any previously sent data will be fully processed. A value of zero indicates that the CICAM does not have any transfer size constraints.

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
for (i=0; i<number_of_metadata_records; i++) {		
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++) {		
drm_metadata_byte	8	uimsbf
}		
}		
}		
if (ts_flag == 0){		
number_of_Sample Tracks	8	uimsbf
for (i=0; i<number_of_Sample_Tracks; i++){		
reserved	3	bslbf
Sample_track_PID	13	uimsbf
number_of_metadata_records	8	uimsbf
for (i=0; i<number_of_metadata_records; i++) {		
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++) {		
drm_metadata_byte	8	uimsbf
}		
}		
}		
}		
}		

The fields are defined as follows:

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 CENELEC 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 Table 34.

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 ETSI 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 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 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. Further details on updating the tracks or the metadata for a session can be found in clause 7.5.3.4.4.

Table 39: sd_update_reply APDU syntax

Syntax	Number of bits	Mnemonic
sd_update_reply() {		
sd_update_reply_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
drm_status	8	uimsbf
}		

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 CENELEC 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

Where 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.

Where 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.

Host player mode TS sample delivery should use a network scrambling algorithm that is capable of encrypting a whole TS packet at once, in a non-divisible way, and should not be vulnerable to attacks that split or re-arrange the TS packets.

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* (see clause 7.5.5.2.1).

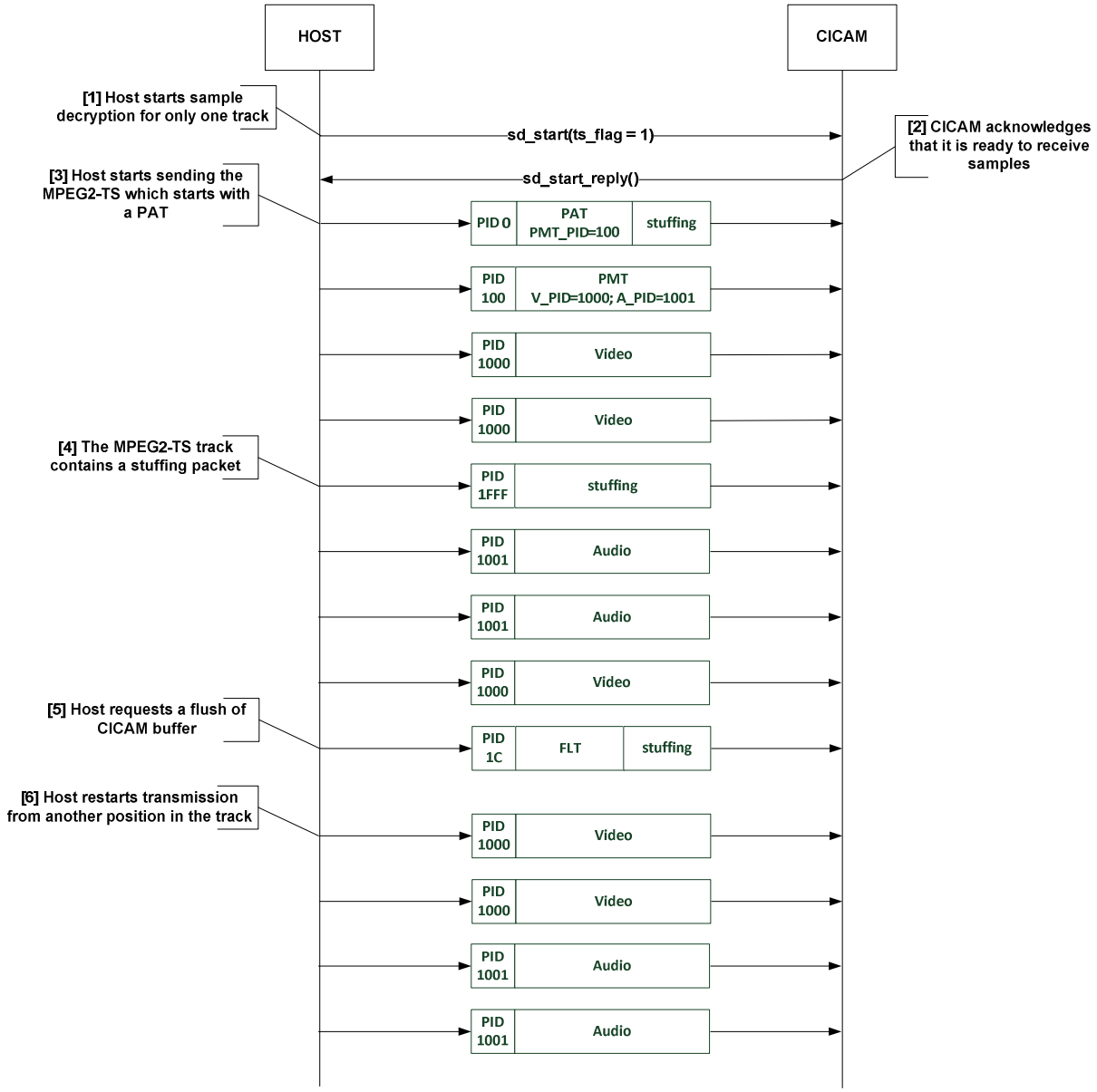

Figure 9: Example sequence with TS content carriage from Host to CICAM

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.

7.5.2.2 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.

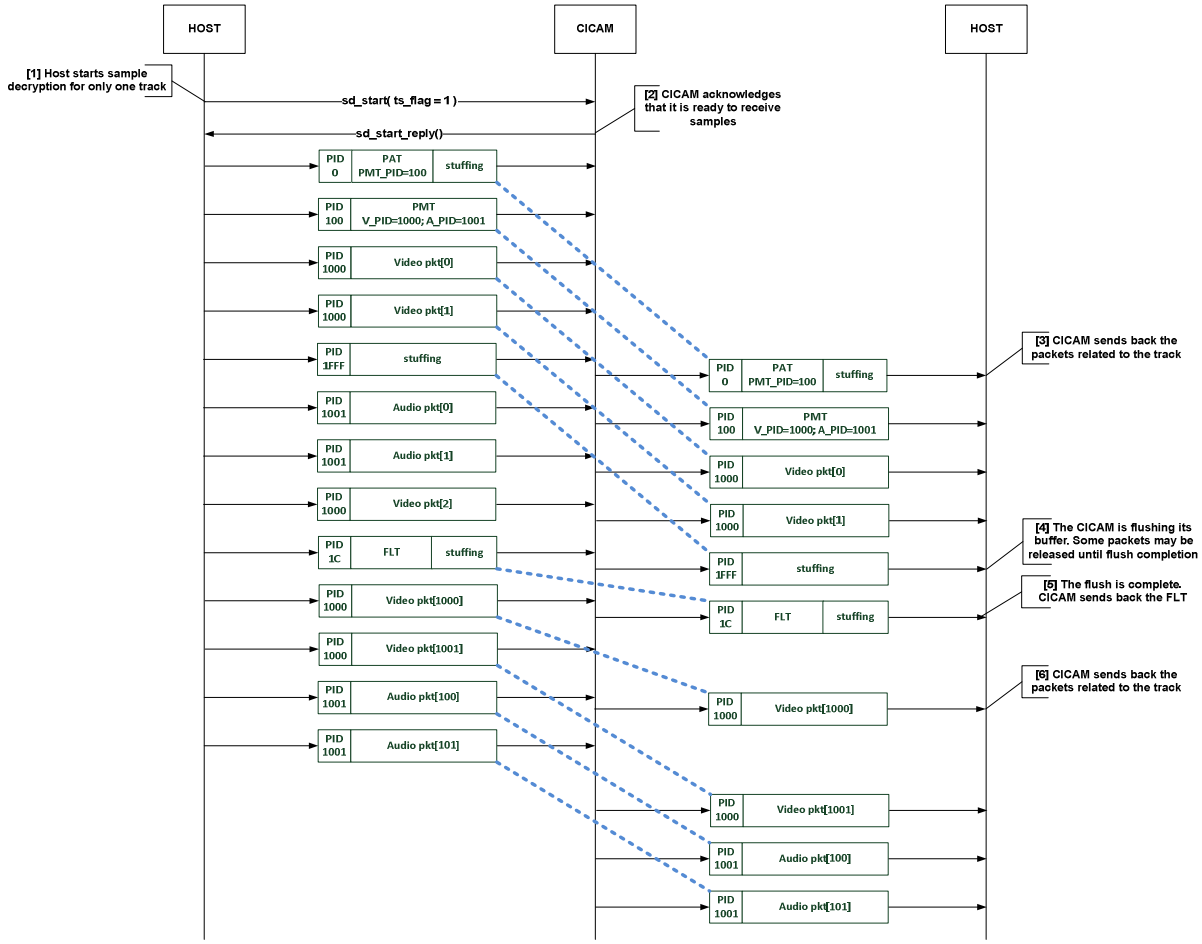

Figure 10: Example sequence with TS content carriage from Host to CICAM and back

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 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.

4) 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.

5) The CICAM has completed the flush and sends back the FLT unmodified as an acknowledgement.

6) The CICAM sends back the packets related to the Track over the TS Interface.

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 General

The CICAM should observe the following rules for CI Plus Link Scrambling control for Host player mode with non-TS content. The exact operation of the CICAM is determined by the network scrambling algorithm which is known by the CICAM:

a) Where the network scrambling algorithm encrypts a whole TS packet at once, in a non-dividable way, and is not vulnerable to attacks that split and re-arrange TS packets ("TS repackaging") then the CICAM may descramble the content and return it to the Host using the CI Plus Link security which includes terminating and solitary short blocks sent in the clear.

b) Where the network scrambling algorithm is vulnerable to TS repackaging then the CICAM should only descramble the content and return it to the Host if the CICAM is able to protect every byte using the CI Plus link security. This may require padding to be added as described in clause 7.5.3.2 to ensure that each TS packet payload is an exact multiple of the cipher block size.

   The CICAM should only descramble TS packets whose payloads are an exact multiple of the CI Plus cipher block size. TS packets whose payload is not an exact multiple of the cipher block size and cannot be padded to be a multiple of the cipher block size as described in clause 7.5.3.2 are returned to the Host with the transport error indicator bit set and their content is still network scrambled (i.e. the payload content has not been descrambled by the CICAM).

NOTE: When using MPEG CENC ISO/IEC 23001-7 [14] pattern mode, the BytesofProtectedData in the clear on the input will be encrypted by the CI Plus link protection at the output.

The recommended operation of the CICAM in Host player mode with non-TS content is shown in the informative Figure 11.

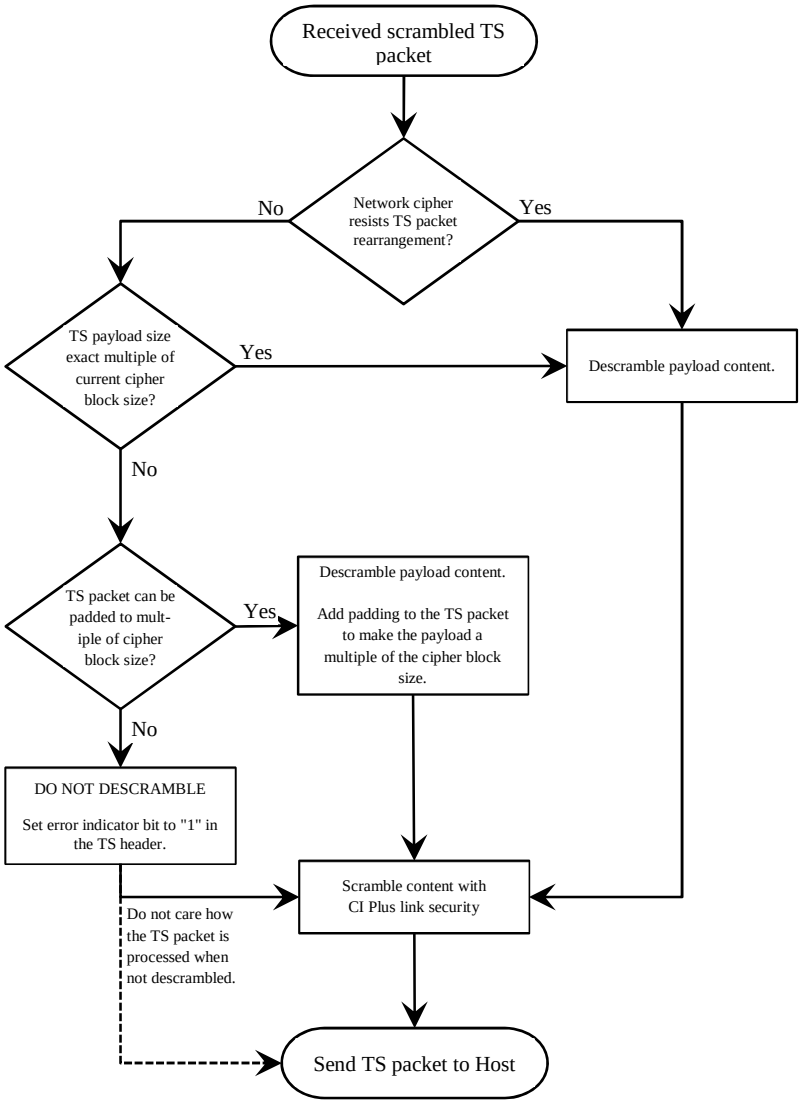

Figure 11: CICAM Operation in Host Player Mode (Informative)

7.5.3.2 TS packet padding

In Host player mode with non-TS content, where the CICAM is required to protect every byte using the CI Plus link security to prevent attacks using the residual block in the clear, the CICAM shall detect scrambled TS packet payloads which are not a multiple of the cipher block size and reformat the TS packet such that it is a multiple of the cipher block size. For the avoidance of doubt this is a TS packet where the **transport_scrambling_control** field of the transport_packet() is defined as '10' or '11' and the payload content **data_byte** size is not a multiple of the CI Plus link protection cipher block size currently negotiated by the CICAM and Host.

To protect every byte of the TS packet payload the packet shall be modified before it is returned to the Host as follows:

a) The TS scrambled payload size shall be increased to be a multiple of the cipher block size. Any new payload bytes shall be inserted before the original payload and originate from the CICAM random number generator.

a) Packet PAD signalling shall be inserted into the private_data_byte as described in clause 7.5.3.3.

b) The adaptation field size shall be reduced by removal of stuffing bytes to allow the TS packet payload to be padded. The CICAM is required to re-author the TS packet construction including the adaptation_size and padding.

Where there is insufficient space in the TS packet to allow a TS packet payload to be padded to a multiple of the cipher block size the CICAM shall not descramble the TS packet and shall set the **transport_error_indicator** bit to "1". In all instances of a valid solitary short block transmission then the adaptation field is filled with stuffing bytes and the TS packet may be adjusted by reducing the **adaptation_field_length** and discarding stuffing bytes to re-arrange the TS packet for the padding and associated signalling. The CICAM shall never allow an unpadded short block to be sent to the Host.A padded short block shall be scrambled by the CICAM using the CI Plus link scrambling mechanism. The CICAM shall ensure that only padded TS packets contain padding signalling in the adaptation field.

A Host receiving the TS from the CICAM shall descramble the TS packets, then detect and discard the padding.

7.5.3.3 TS packet PAD signalling

The TS packet padding signalling conveys information about the TS packet. The PAD signalling is included in the first 4 bytes of the adaptation_field transport_private_data bytes. The adaptation_field shall be modified as follows:

i) The **transport_private_data_flag** field shall be set to "1".

ii) The **transport_private_data_length** field shall be set to 4 bytes.

The adaptation padding marker carried in the transport_private_data field is defined as follows.

Table 40: Adaptation Padding Marker

Syntax	Number of bits	Mnemonic
`adaptation_padding_marker {`		
`  pad_indicator`	24	uimsbf
`  reserved_zero`	2	bslbf
`  padding_length`	6	uimsbf
`}`		

Where the fields of the adaptation_padding_marker are defined as follows:

pad_indicator: This 24-bit field identifies the start of the padding indicator. This field shall have a value of 0x504144 ("PAD").

reserved_zero: Reserved for future use and shall be zero. Receivers shall be tolerant to this field containing any other value and shall not interpret the field.

padding_length: This 6-bit field indicates the amount of padding in bytes that has been applied to the TS packet payload. The value of this field may be zero which indicates that there is no padding applied.

The padding mechanism is depicted in Figure 12.

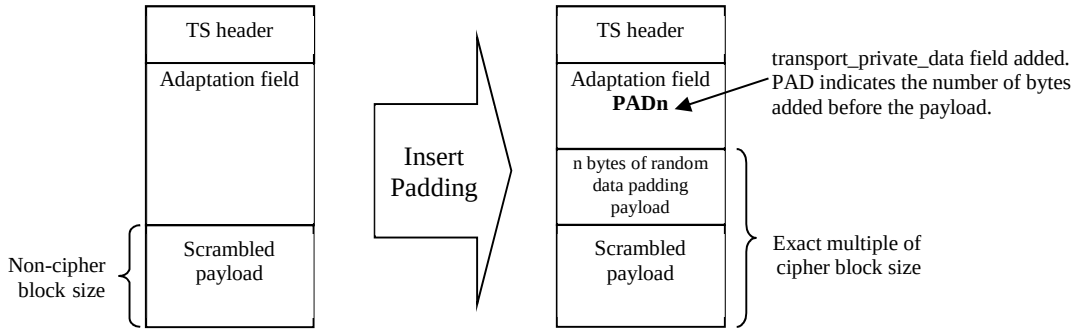

Figure 12: TS packet padding

A Host receiving a padded TS packet first decrypts the payload, and then discards the padding by adding the value of the padding_length field to the adaptation_field_length of the TS packet.

7.5.3.4 Host output

7.5.3.4.1 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 only contain TS packets only with the following PIDs:

- the comms PID;
- 0x1FFF (NULL packets);
- those declared in the *sd_start()* or *sd_update()* APDUs.

The CICAM shall ignore NULL packets and pass them through.

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.

The Host shall send non-TS content carriage data in TS packets of a payload size that is an exact multiple of CI Plus link protection cipher block size units such that the TS packet shall not include a short block when scrambled on the CI Plus link. All TS packets for an encrypted block of sample data, except for the last TS packet for that block, shall always be filled up to the maximum integer number of cipher block sizes that can fit in the payload.

Where it is not possible to create a payload that is an exact multiple of the CI Plus link protection cipher block size the Host shall ensure that TS packet containing the short block includes sufficient space in the adaptation field for the CICAM to pad the TS packet according to clause 7.5.3.3.

7.5.3.4.2 Transmitting Samples

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

The *transport_scrambling_control* bits of the SSP indicate the parity of the *transport_scrambling_control* bits of the transport packets containing encrypted data of the announced Sample. The Host shall toggle the *transport_scrambling_control* bits in the SSP of the next Sample of a Sample Track, alternating between an odd and even parity.

After the SSP 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 SEP before sending the SSP of the next Sample for the same Track.

After transmission of the transport packet containing the end of a Sample, the Host shall send a Sample End TS Packet (SEP) 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 SEP for that Sample. Where the *sd_start_reply()* data_block_size field was specified as greater than zero, the Host may choose to send data_block_size NULL TS packets to ensure that the Sample data is processed by the CICAM.

The sequence diagram in Figure 13 illustrates informatively the usage of the SSP and SEP by the Host.

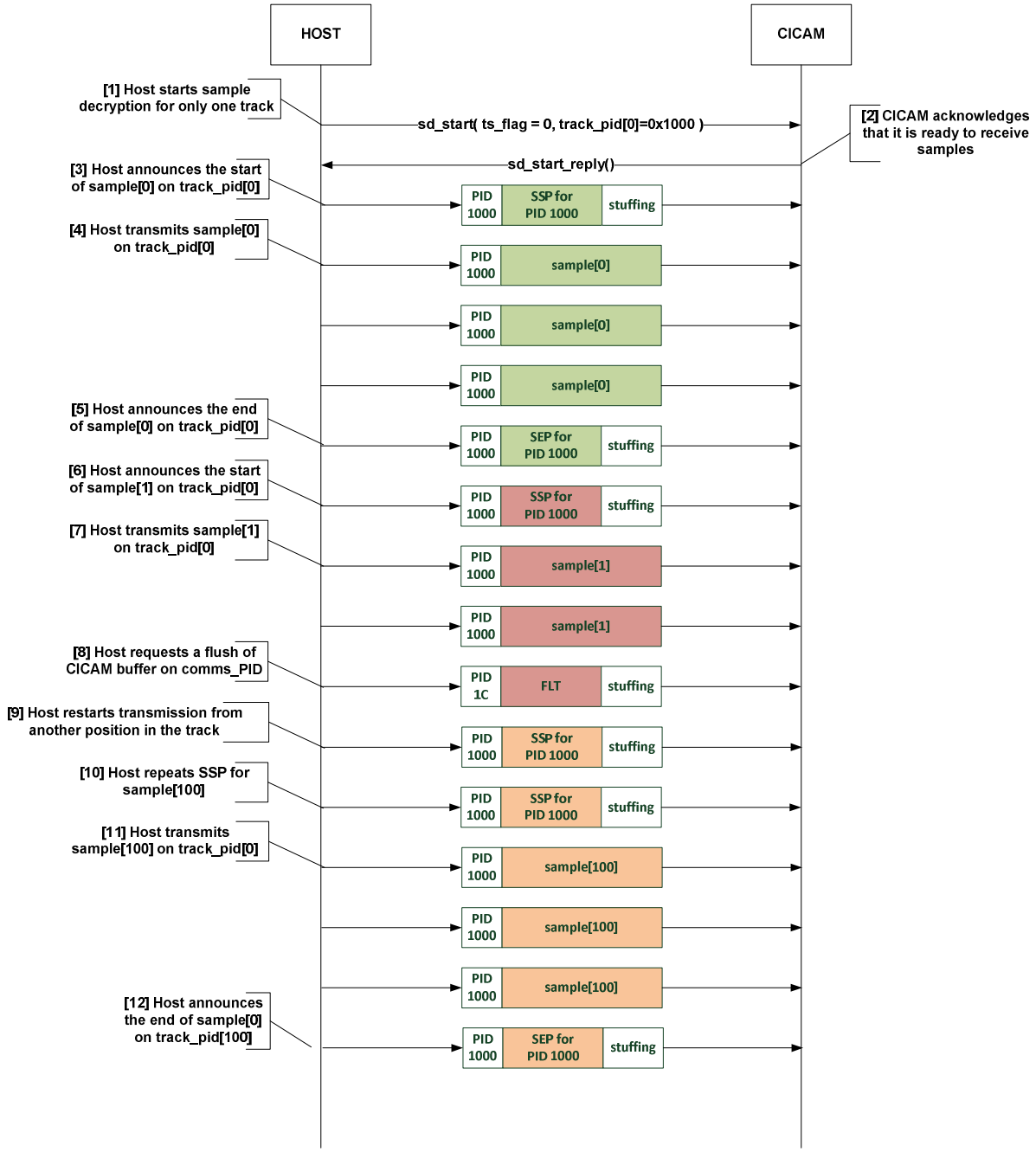

Figure 13: Example sequence with non-TS content carriage from Host to CICAM

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 SSP 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 SEP on the *track_PID*.

6) The Host announces the start of the Sample[1] by sending a SSP 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 SSP on *track_PID*. The Host is very likely performing a jump operation.

10) Host repeat the SSP 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 SEP 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 an SSP.

7.5.3.4.3 Managing multiple Tracks

When more than one Track is declared, the Host shall not interleave Samples originating from different Tracks. The Host is required to send the complete Sample data of a Track before sending the next Sample of another Track. This restriction minimizes the buffering requirements of the CICAM and simplifies the send and receive mechanism of the Host.

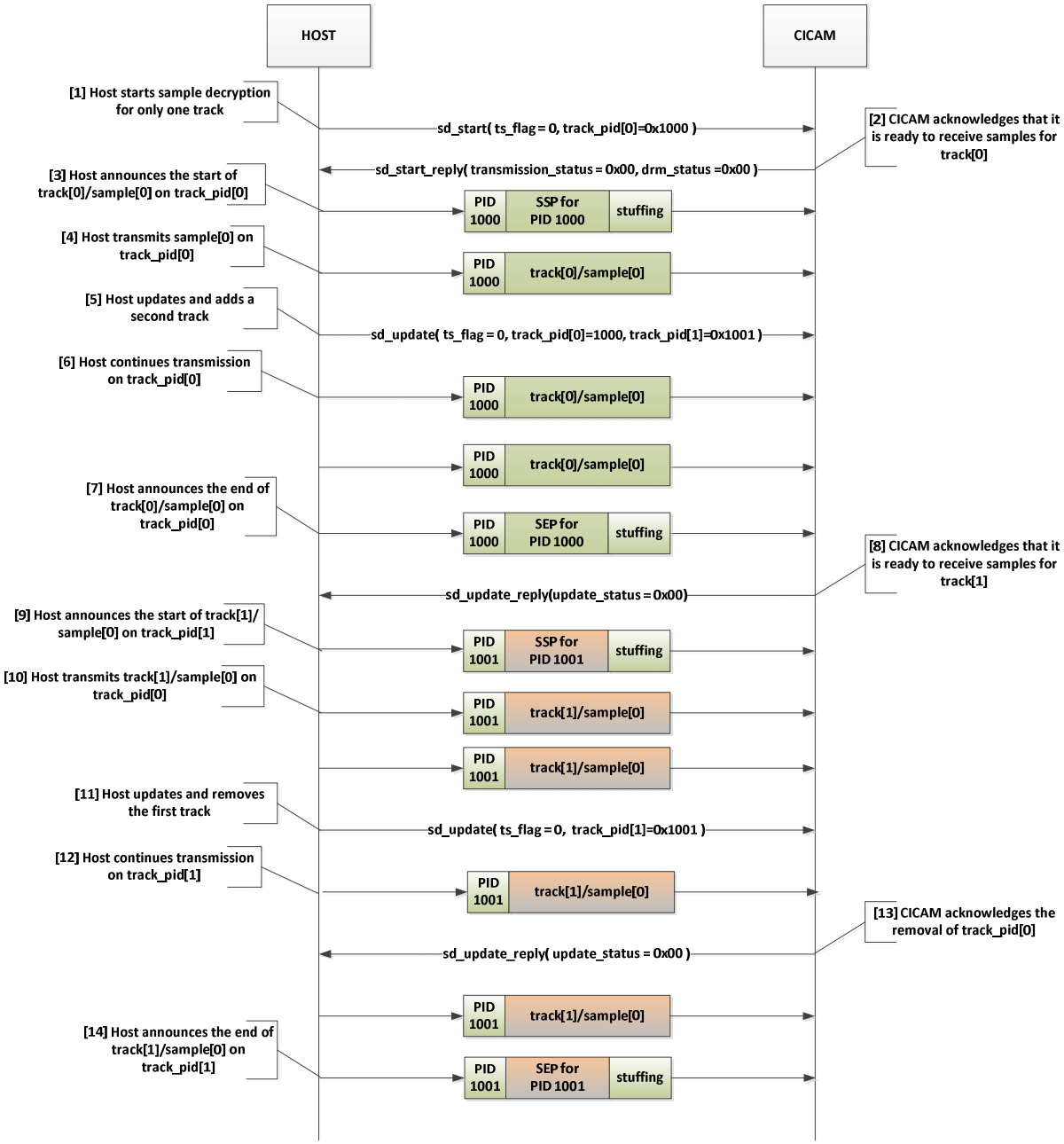

Figure 14: Example sequence with multiple tracks on non-TS content carriage from Host to CICAM

NOTE: All PID numbers are hexadecimal.

1) The Host starts Sample decryption over the TS Interface by sending 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 SSP 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 SSP 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 SEP on the *track_PID*.

8) Host announces the start of the Track[1]/Sample[1] by sending a SSP 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.

7.5.3.4.4 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 host shall not update the track list before the setting up of the playback session has been finalized and confirmed by the CICAM with an *sd_start_reply()* APDU conveying a DRM status other than "undetermined". For subsequent track list updates, the host shall wait for an *sd_update_reply()* confirming that the CICAM has applied the previous track list update, before sending the next *sd_update()* APDU. The sequence diagram in Figure 15 illustrates the Host output when updating the Track list.

NOTE: When a Host sends data from various sources, such as e.g. when playing back a playlist, resulting in use of several, different keys, which may result in the same key_id being used in different assets but with a different key value, then the Host should make sure that all data from the previous key has been processed by the CICAM before sending new keys and the associated sample data.

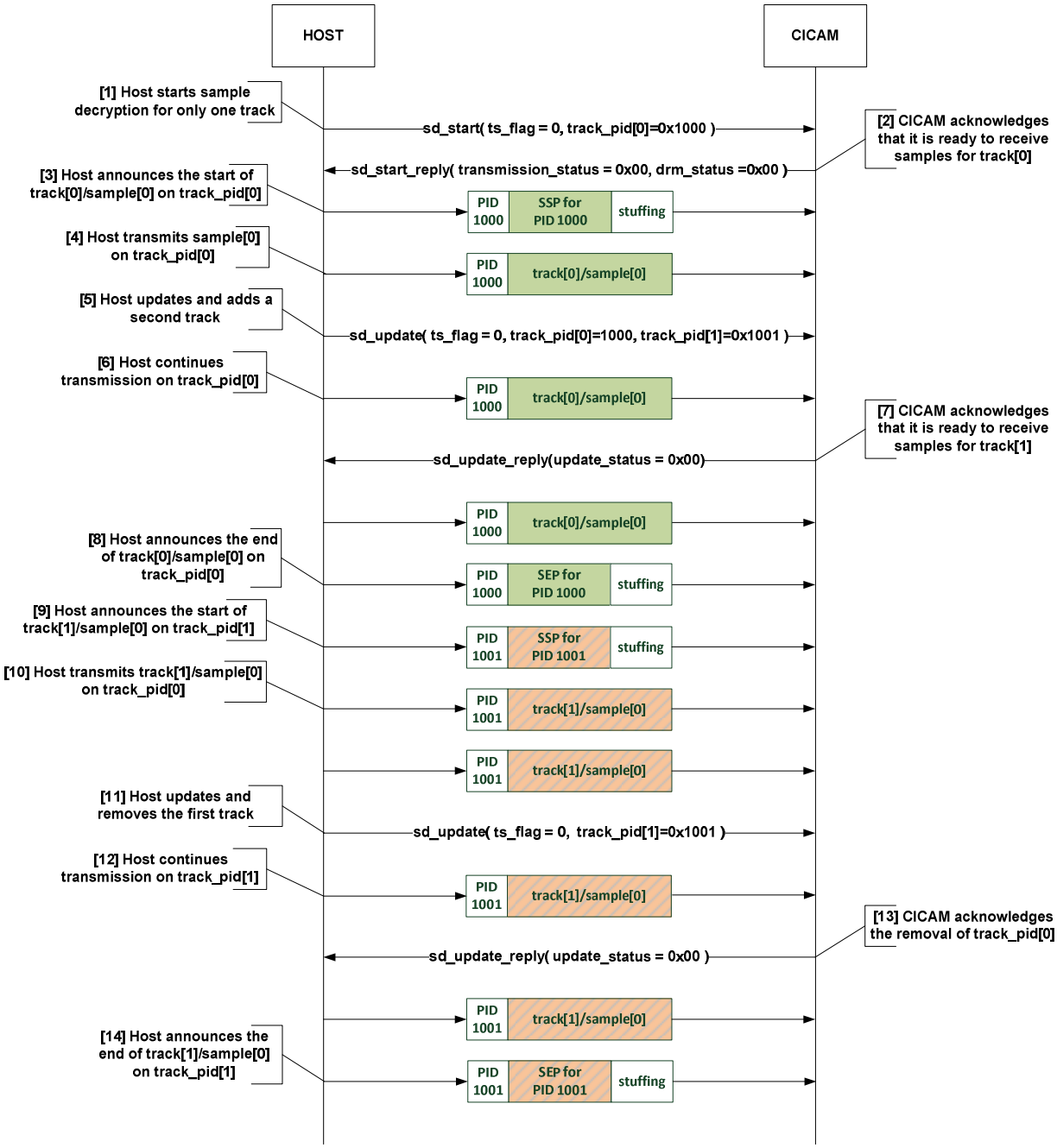

Figure 15: Example sequence with non-TS content Track list update

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 SSP 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) Transmission of Track[0]/Sample[0] is complete. The Host announces the end of the Sample by sending a SEP on the *track_PID*.

9) Host announces the start of the Track[1]/Sample[0] by sending a SSP on *track_PID*.

10) Host transmits the Track[1]/Sample[0] on track_PID[1] = 0x1001.

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 SEP on the *track_PID*.

7.5.3.5 TS packets

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* shall be set to 0b0.
- 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 *transport_scrambling_control* indicated in the SSP of the corresponding sample, as defined in clause 7.5.5.5, 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 set according to MPEG-2 Systems [4], including indication of discontinuities.

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;

- a TS packet that contains encrypted data when it is filled up to the maximum integer number of cipher block sizes that can fit in the payload.

7.5.3.6 CICAM output

The relative order of transport packets shall remain in the same order as sent by the Host, and shall follow the rules in clause 7.5.3.2 for setting the TS packet header fields.

NOTE 1: The CICAM may regenerate the TS packet headers, hence the Host should not rely on them being identical when received back.

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*.

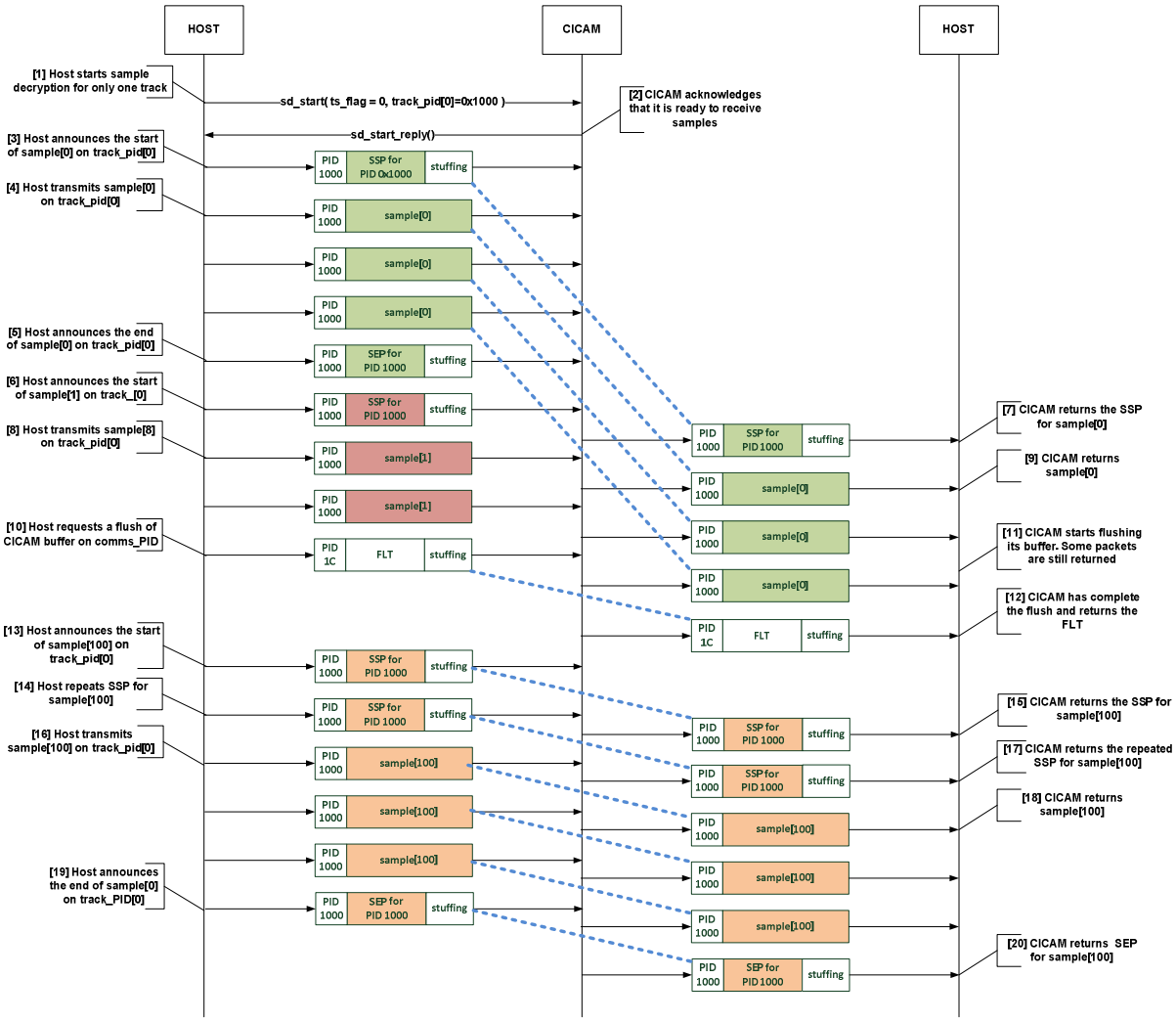

Figure 16: Example sequence with non-TS content buffer flush

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 SSP 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 SEP on the *track_PID*.

6) Host announces the start of the Sample[1] by sending a SSP on the *track_PID*.

7) CICAM returns the SSP 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 SSP on the *track_PID*, before it receives the flush confirmation from the CICAM.

14) Host repeats the SSP.

15) CICAM returns the SSP.

16) Host starts the transmission of Sample[100].

17) CICAM returns the repeated SSP.

18) CICAM returns Sample[100].

19) Host announces the end of transmission of Sample[100] by sending a SEP on the *track_PID*.

20) CICAM returns the SEP on the *track_PID* after the last packet of Sample[100] has been returned.

7.5.3.7 ISOBMFF Samples

7.5.3.7.1 Sample Start TS Packet (SSP)

For the announcement of a clear Sample originating from an ISOBMFF file, the descriptor loop of the SSP (see clause 7.5.5.3.2) 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 SSP.

The `initialization_vector_descriptor()` shall contain the InitializationVector of the Sample, as defined in ISO/IEC 23001-7 [14] in the *iv_data* bytes.

The `key_identifier_descriptor()` shall contain the KID of the Sample, as defined in ISO/IEC 23001-7 [14] in the key_id_data bytes.

7.5.3.7.2 ISOBMFF Sample packetization

Packaging of ISOBMFF Samples into a TS shall follow the rules as described above and as illustrated in Figure 17.

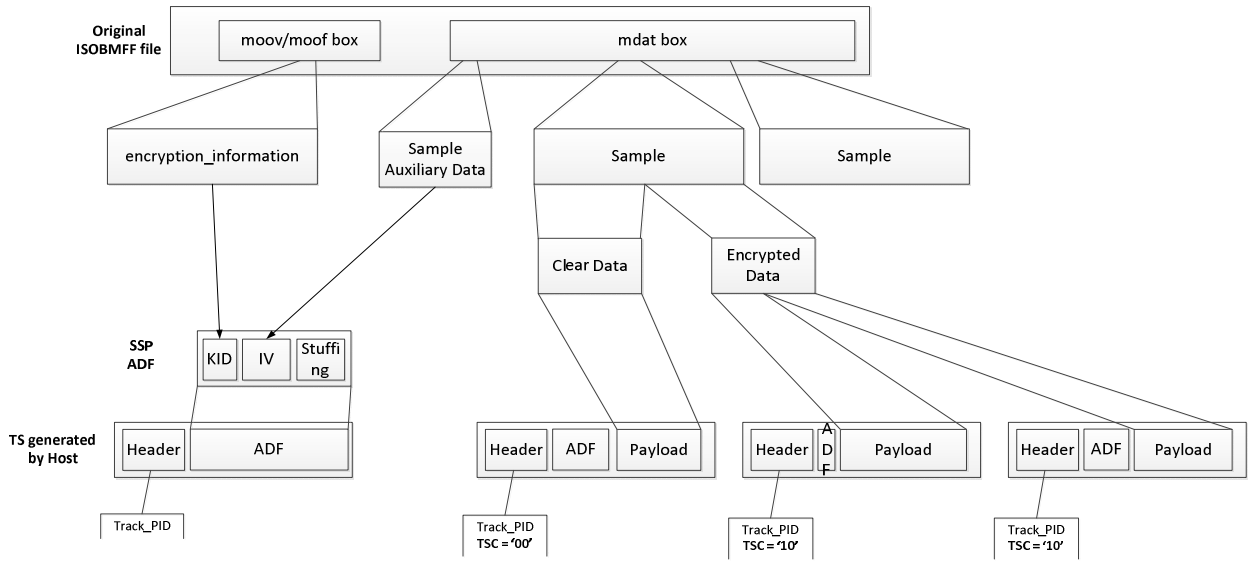

Figure 17: Encapsulation of ISOBMFF content into a TS

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 Local TS by use of the *sd_start_reply()* APDU.

7.5.4.3 Buffer level management

When sending Sample data to the CICAM, the Host shall manage the CICAM buffer level fullness for each Local TS by keeping a record of the number of TS packets sent and received. By comparing these packet sent and received counts to the total buffer_size reported by the CICAM in the *sd_start_reply()* APDU (see clause 7.4.5), the Host can determine the instantaneous CICAM buffer fullness. The Host shall not make any assumptions on buffer draining in the CICAM and shall not send more than the last known available space as computed by the Host.

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.

The Host shall not assume that the flush has been completed and that the CICAM buffer has been freed until it receives the FLT back from the CICAM.

7.5.5 Messages and descriptors

7.5.5.1 Introduction

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

7.5.5.2 General messages

7.5.5.2.1 General

Messages that are not related to a track (currently only the FLT) shall be mapped as MPEG2 private sections [4] directly into TS packets.

Stuffing shall be performed by filling each remaining byte of the TS packet with the 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 message section, the following 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 *transport_priority* shall be 0b0.
- The *PID* shall be set to the *comms_PID* value of 0x001C.
- The *transport_scrambling_control* shall be set to 0b00: messages 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 message section.

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

7.5.5.2.2 Coding of table_id values

Table 41 lists the values which shall be used for *table_id* for the general messages, defined in the present document.

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

Table 41: General messages table_id values

Table Identifier	Description
0x00 to 0xD1	reserved
0xD2	flush_section
0xD3 to 0xFE	reserved
0xFF	forbidden

7.5.5.2.3 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 42.

Table 42: Flush section syntax

Syntax	Number 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; i<N; 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.

7.5.5.3 Track-related messages

7.5.5.3.1 General

Messages that are related to a track, i.e. SSP and SEP are mapped as Transport Stream Packets as defined in MPEG-2 Systems [4]. In this way, it is possible to transmit the beginning (SSP) and end (SEP) signalling in the same PID stream as the Samples themselves.

For a TS packet being a track-related message, 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 *transport_priority* shall be 0b0.
- The *PID* shall be the same as the track_PID it is related to.
- 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].

NOTE 1: As the TS packet contains only an adaptation field, the adaptation field length is set equal to 183 (bytes).

The contents of the Adaptation field shall be:

- 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 private data of a given packet shall contain all descriptors of an entire track-related message. A track-related message shall not be split over multiple TS packets.

NOTE 2: A track-related message may not be larger than a single TS packet.

The *private_data_bytes* contained in an adaption field may contain the following descriptors from those specified in the present document and shown in Table 45, and from those defined in CI Plus V1.3 [3]:

- *ciplus_initialization_vector_descriptor*.
- *ciplus_key_identifier_descriptor*.

7.5.5.3.2 Sample Start TS Packet (SSP)

The Sample Start TS Packet (SSP) is a track-related message as defined in clause 7.5.5.3.1. The SSP indicates the start of a Sample. For setting the TS header fields the following rules shall apply:

- The *payload_unit_start_indicator* of a SSP shall be set to 0b1.
- The *transport_scrambling_control* shall be set to the odd, even, or clear value for the next Sample. The Host shall alternate between even and odd between encrypted Samples of a track.

NOTE: This means that if one encrypted Sample of a track uses the even *transport_scrambling_control* value, the next encrypted Sample of that track will use the odd value, and the next encrypted Sample after that will again use the even value, and so on.

For setting the adaptation field the following rules shall apply:

- Transport_private_data_length: Size of the sample_start_transport_private_data (see Table 43).

The private_data bytes shall be encoded according to Table 43.

Table 43: SSP syntax

Syntax	Number of bits	Mnemonic

`sample_start_transport_private_data(){`
`  for (i=0; i<N; i++) {`
`  descriptor()`
`  }`
`}`		

The fields are defined as follows:

descriptor: The descriptors within the SSP, 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.

7.5.5.3.3 Sample End TS Packet (SEP)

The Sample End TS Packet (SEP) is a track-related message as defined in clause 7.5.5.3.1. The SEP indicates the end of a Sample. For setting the TS header fields the following rules shall apply:

- The *payload_unit_start_indicator* of a SEP shall be set to 0b0.
- The *transport_scrambling_control* shall be set to 0b00.

For setting the adaptation field the following rules shall apply:

- Transport_private_data_length: Size of the sample_end_transport_private_data (see Table 44).

The transport_private_data bytes shall be encoded according to Table 44.

Table 44: SEP syntax

Syntax	Number of bits	Mnemonic

`sample_end_transport_private_data(){`
 	
`  for (i=0; i<N; i++) {`
`  descriptor()`
`  }`
`}`		

The fields are defined as follows:

descriptor: The descriptors within the SEP, 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.

7.5.5.4 Descriptors

7.5.5.4.1 General

This clause defines the descriptors that may be used within Sample Mode messages. Zero or more of these descriptors may be inserted into any of the messages specified in clauses 7.5.5.2 and 7.5.5.3, 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 Sample Mode messages. A " * " character in the intersection between a descriptor and a message type indicates that the descriptor may be included in that message type.

Table 45: Message descriptors

Descriptor	Tag Value	SSP	SEP	FLT
Forbidden	0x00			
ciplus_initialization_vector_descriptor	0xD0	*		
ciplus_key_identifier_descriptor	0xD1	*		
Reserved	0xD2 to 0xEF			
Host defined	0xF0 to 0xFE	*	*	*
Forbidden	0xFF			

7.5.5.4.2 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
ciplus_initialization_vector_descriptor(){		
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_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.

7.5.5.4.3 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
ciplus_key_identifier_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0; i<N; i++) {		
key_id_data_byte	8	uimsbf
}		
}		

The fields are defined as follows:

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

7.6 URI

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

8 IP delivery CICAM player mode

8.1 General

The CI Plus 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 Plus 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 8.8 to inform the CICAM about user actions.

8.3 Session initialization

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 Plus 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 Plus 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_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 for this Local TS 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 or application MMI message, 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 by 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 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.

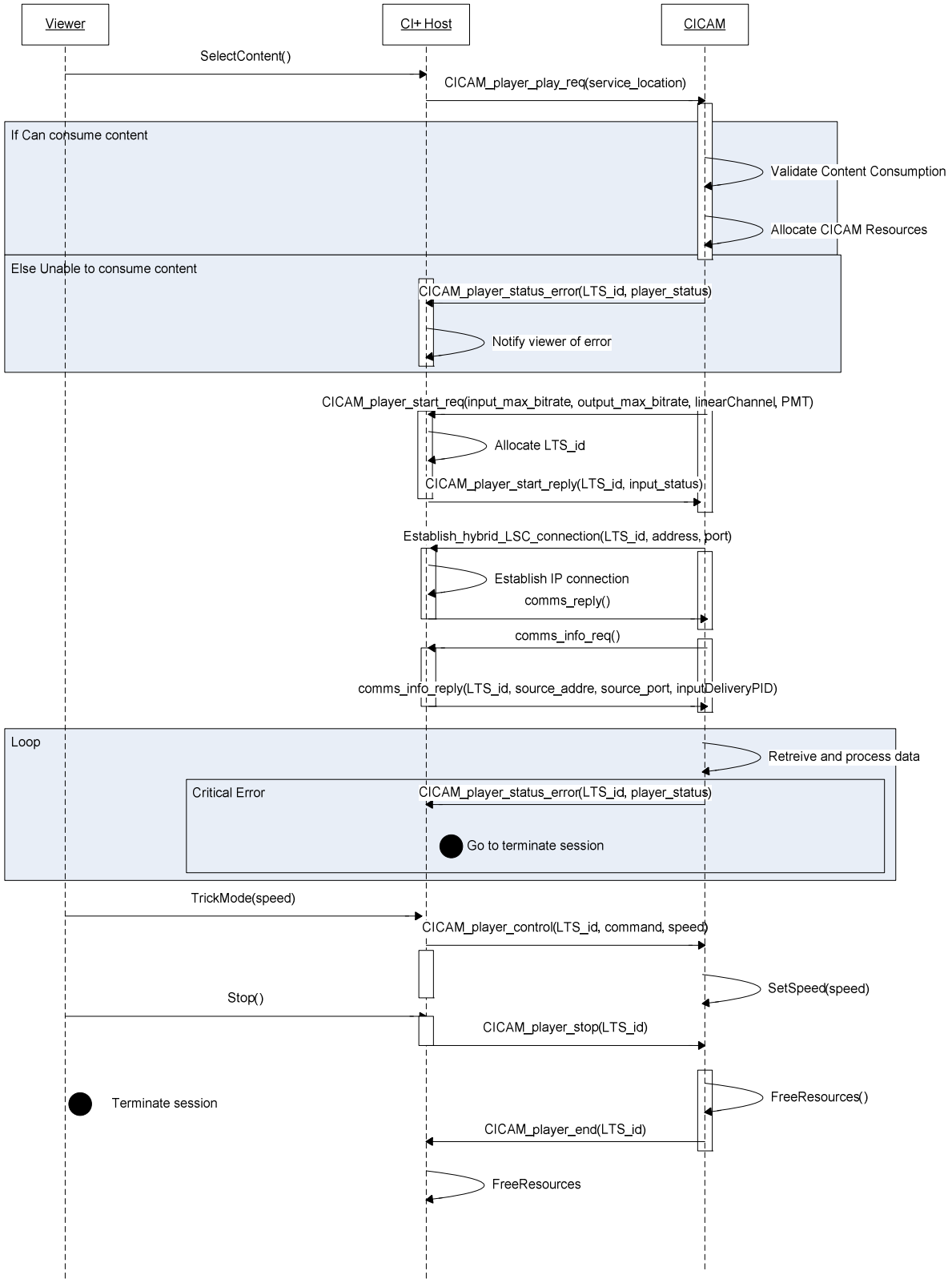

Figure 18: Example sequence diagram for Host-initiated playback

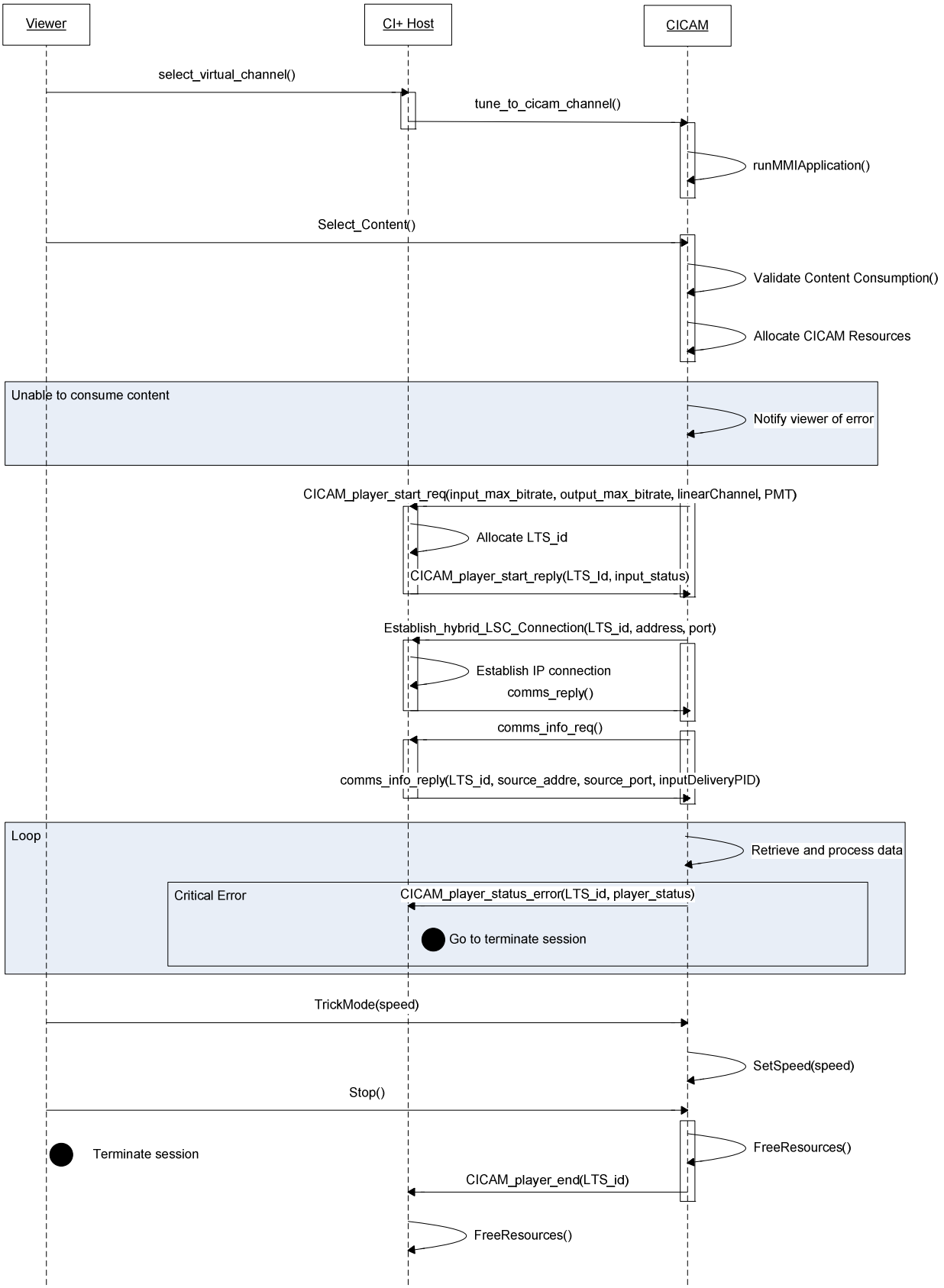

Figure 19: Example sequence diagram for CICAM-initiated playback

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.

In order to allow the Host to request the start of player session, the CICAM should keep a session open at all times.

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 Plus 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.

Table 48: CICAM Player resource APDUs

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

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
CICAM_player_verify_req() {		
CICAM_player_verify_req_tag	24	uimsbf
length_field()		
service_location_length	16	uimsbf
for (i=0; i<service_location_length; i++) {		
service_location_byte	8	
}		
}		

The fields are defined as follows:

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	Number of bits	Mnemonic
CICAM_player_verify_reply() {		
CICAM_player_verify_reply_tag	24	uimsbf
length_field()= 1		
player_verify_status	8	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.

If the CICAM needs to know if a specific combination of components can be played by the Host, it shall use *CICAM_player_codec_verify_req()* APDU as defined from CICAM Player version 2 (see clause 18.2).

The syntax of the *CICAM_player_capabilities_req()* APDU is shown in Table 52.

Table 52: CICAM_player_capabilities_req APDU syntax

Syntax	Number of bits	Mnemonic

CICAM_player_capabilities_req() {
  CICAM_player_capabilities_req_tag
  length_field()= 0
}
	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
  length_field()
  number_of_component_types
  for (i=0; i<number_of_component_types; i++) {
  stream_content
  reserved
  component_type
  }
}
	
	24
	
	16
	
	4
	4
	8
		
		uimsbf
		
		uimsbf
		
		uimsbf
		bslbf
		uimsbf

The fields are defined as follows:

CICAM_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 ETSI 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 ETSI 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
CICAM_player_start_req() {		
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++) {		
PMT_byte	8	bslbf
}		
}		

The fields are defined as follows:

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. This is the bit rate reserved on the TS interface for delivering data from Host to CICAM for this player session.

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. This is the bit rate reserved on the TS interface for delivering data from CICAM to Host for this player session.

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 remaining 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
CICAM_player_start_reply() {		
CICAM_player_start_reply_tag	24	uimsbf
length_field()= 2		
LTS_id	8	uimsbf
input_status	8	uimsbf
}		

The fields are defined as follows:

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	Number of of bits	Mnemonic
CICAM_player_play_req() {		
CICAM_player_play_req_tag	24	uimsbf
length_field()		
service_location_length	16	uimsbf
for (i=0;i<service_location_length;i++) {		
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	Number of bits	Mnemonic
CICAM_player_status_error() {		
CICAM_player_status_error_tag	24	uimsbf
length_field() = 3		
reserved	7	uimsbf
valid_LTS_id	1	uimsbf
LTS_id	8	
player_status	8	
}		

The fields are defined as follows:

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. unsupported content format or protocol)	0x01
Error - unrecoverable error	0x02
Error - content blocked (e.g. no content license available)	0x03
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 actual 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
if(command == 0x01) {		
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 0xFFFFFFFF 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 means × 2 in forward direction
- speed = 50 means × 0,5 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

CICAM_player_info_req() {
  CICAM_player_info_req_tag
  length_field() = 1
  LTS_id
}
	
	24
	
	8
		
		Uimsbf
		
		uimsbf

The fields are defined as follows:

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	Number of bits	Mnemonic

CICAM_player_info_reply() {
  CICAM_player_info_reply_tag
  length_field() = 11
  LTS_id
  duration
  position
  speed
}
	
	24
	
	8
	32
	32
	16
		
		uimsbf
		
		uimsbf
		uimsbf
		uimsbf
		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 0xFFFFFFFF.

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 0xFFFFFFFF.

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	Number of bits	Mnemonic
CICAM_player_stop() {		
CICAM_player_stop_tag	24	uimsbf
length_field() = 1		
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_end APDU syntax

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

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_asset_end_tag	24	uimsbf
length_field() = 2		
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 identifier 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() {		
CICAM_player_update_req_tag	24	uimsbf
length_field()		
LTS_id	8	uimsbf
PMT_length	16	uimsbf
for(int i=0; i<PMT_length; i++) {		
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
CICAM_player_update_reply() {		
CICAM_player_start_reply_tag	24	uimsbf
length_field()= 2		
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 is ready to receive the Local TS	0x00
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**	**Type**	**Ver.**	**APDU tag**	**Tag value**	**Host**	**CICAM**
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	→	
					CICAM_player_start_req	9F A0 04	←	
					CICAM_player_start_reply	9F A0 05	→	
					CICAM_player_play_req	9F A0 06	→	
					CICAM_player_status_error	9F A0 07	←	
					CICAM_player_control_req	9F A0 08	→	
					CICAM_player_info_req	9F A0 09	→	
					CICAM_player_info_reply	9F A0 0A	←	
					CICAM_player_stop	9F A0 0B	→	
					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	→	

9 CICAM file retrieval

9.1 General

CICAM file retrieval is realized via a new CI Plus 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 Plus 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 one session for each DomainIdentifier it supports.

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.

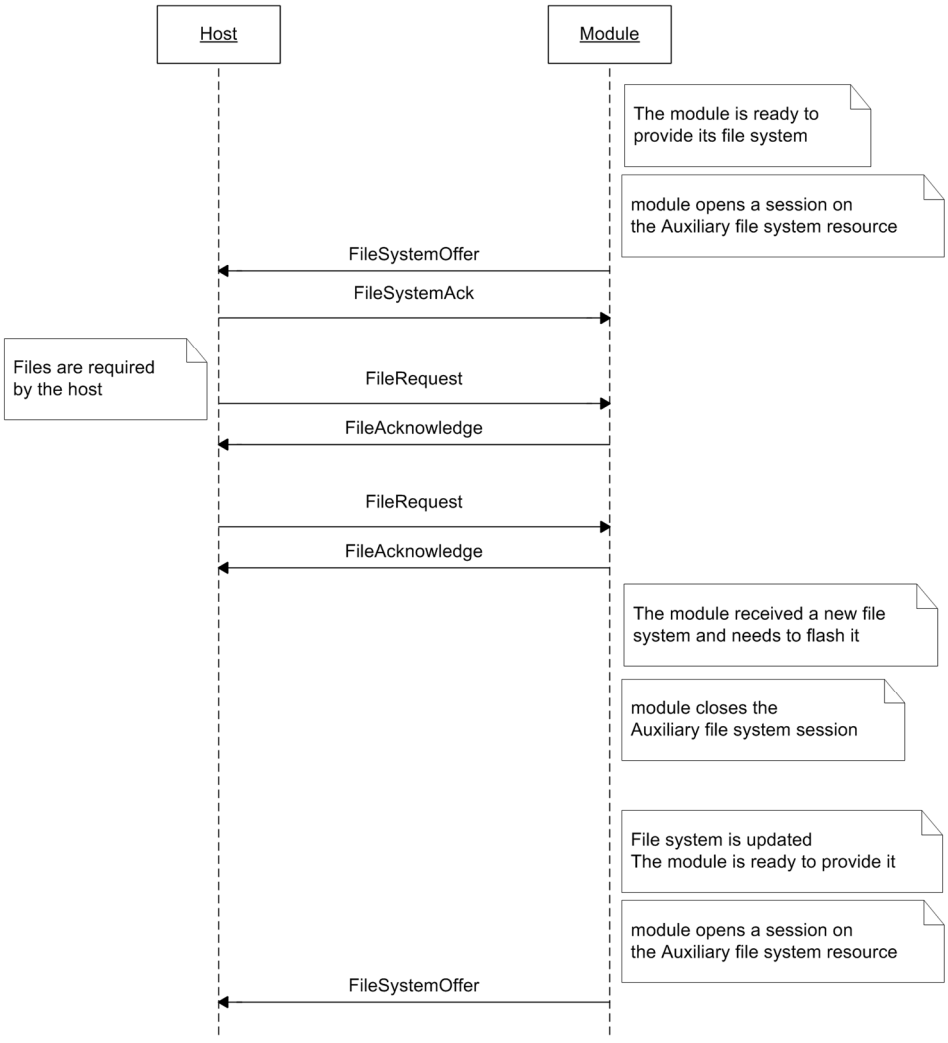

Figure 20: Example usage of the auxiliary file system resource

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 CENELEC 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 organization 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 IETF 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	Number 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 CENELEC 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	Unknown DomainIdentifier. The DomainIdentifier is not supported 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 Plus V1.3 [3].

It allows the Host to support file caching and to discover reqtypes supported by the CICAM (file, data, hash, etc.).

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 Plus V1.3 [3].

9.6 Auxiliary file system resource summary

Table 75 provides a summary of the attributes of the auxiliary file system resource.

Table 75: Auxiliary file system resource summary

Resource					Application Object		Direction	
Name	**Resource Identifier**	**Class**	**Type**	**Vers.**	**APDU Tag**	**Tag value**	**Host**	**CICAM**
Auxiliary file system	00 91 00 41	145	1	1	FileSystemOffer	9F 94 00	←	
					FileSystemAck	9F 94 01	→	
					FileRequest	9F 94 02	→	
					FileAcknowledge	9F 94 03	←	

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 Plus 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_IP_config()* APDU to transfer information about the IP network adapters of the Host to the CICAM.
- A new field source_port (in the connection_descriptor) that allows the CICAM to send out a message reusing the UDP or TCP source port of another 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 Plus 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:

- IETF RFC 768 [17] (UDP)
- IETF RFC 793 [19] (TCP)
- IETF RFC 791[18] (IPv4)
- IETF RFC 3376 [20] (IGMPv3)
- IETF RFC 1112 [21] (IP multicast)

For this version of the resource, support for IPv6 is optional for the Host. IPv6 support on the Host shall be compliant with IETF RFC 2460 [22] (IPv6), IETF RFC 4443 [23] (ICMPv6) and IETF RFC 4291 [28] (IP Version 6 Addressing Architecture).

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 TCP or UDP payload 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 initialization, such as RTSP. Further, it may be used when another connection is set up that wants to re-use the same 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 10.11.3), 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 Source ports

For some IP connections, it may be important that the outgoing data uses the same 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 a 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 TCP or UDP payload 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].

When receiving UDP data it is possible that an overflow will occur on the Host side. In that case the Host may drop full UDP packets. The Host shall not drop part of a UDP packet.

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 Plus 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. This is the field *input_max_bitrate* as provided by the CICAM to the Host in the *CICAM_player_start_req()* APDU defined in clause 8.8.7.

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 TCP or UDP payloadwill 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	Mnemonic

comms_info_req () {
  comms_info_req_tag
  length_field() = 1
}
	24	uimsbf

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 CENELEC 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
  length_field() = 22
  LTS_id
  reserved
  status
  source_IPaddress
  source_port
  reserved
  inputDeliveryPID
}
	24
	
	8
	7
	1
	128
	16
	3
	13
		uimsbf
		
		uimsbf
		bslbf
		bslbf
		uimsbf
		uimbsf
		bslbf
		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 CENELEC 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. Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

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 TCP or UDP payloadacross 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_IP_config

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
  length_field() = 0
]
	24	uimsbf

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 CENELEC 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++) {		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 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. Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

network_mask: The subnet mask for this IP network adapter. Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

default_gateway: This field is the IP address of the default gateway used for this IP network adapter. Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

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. Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

num_DNS_servers: The number of DNS servers used for this IP network adapter. Addresses are expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

DNS_server_address: The IP address of the DNS server . Address is expressed in IPV6 format. For an IPv4 address shall be prefixed with ::ffff:0:0/96 as defined in clause 2.5.5.2 of IETF RFC 4291 [28] or ::0:0/96 as defined in clause 2.5.5.1 of IETF RFC 4291 [28].

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) {		
source_port	16	uimsbf
}		
if (connection_descriptor_type == SI_telephone_descriptor) {		
telephone_descriptor ()		
}		
if (connection_descriptor_type == cable_return_channel_descriptor) {		
channel_id	8	uimsbf
}		
if (connection_descriptor_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 == multicast_descriptor) {		
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 a *source_port* is specified.

NOTE: The allocation of the MSB of the *connection_descriptor_type* for the *sourcePortFlag* maintains backward-compatibility with the CI Plus V1.3 [3] connection descriptor.

source_port: The 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 a *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 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	Number 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()		
}		
if (IP_connection_type == hostname_descriptor {		
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 TCP or UDP payloadshall 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 Plus 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++) {		
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.

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

comms_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 Plus 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 TCP or UDP payloadin 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 =='11') {		
adaptation_field()		
}		
for(i=0; i<N; i++) {		
data_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: TS packets containing the first byte of UDP payload shall set it as "1". Other packets shall set this field to "0".

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 field shall be set to 0b00.

data_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.

stuffing_byte: This shall be set to 0xFF.

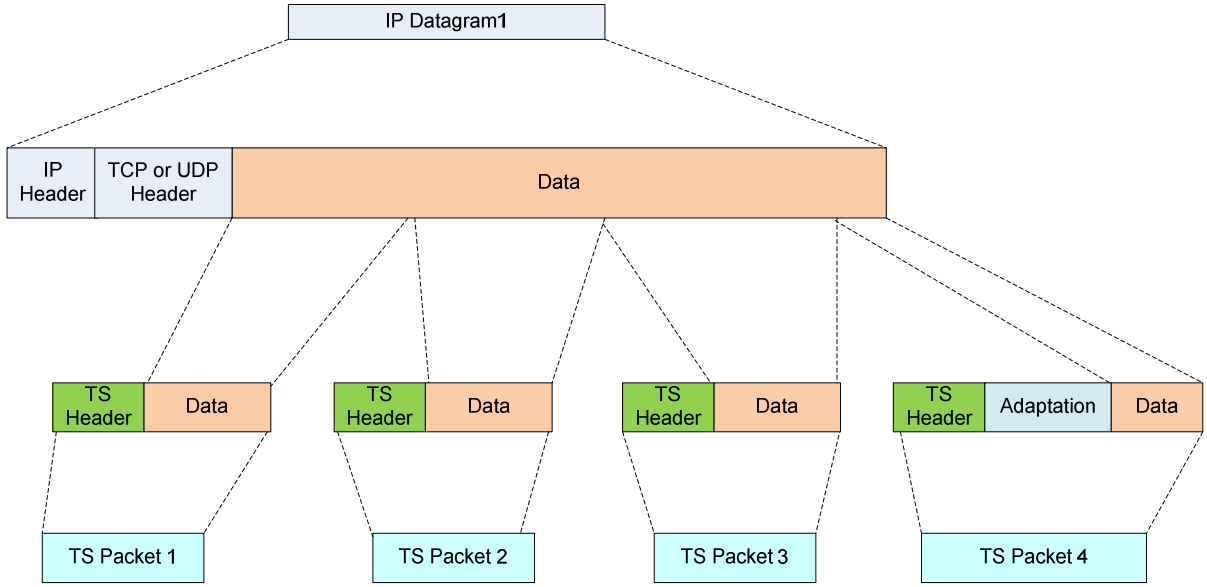

Figure 21: Mapping of IP datagram to TS packets

One TS packet shall only contain the bytes from the same UDP packet. That is to say, in the case where a further IP datagram is ready for delivery, the last TS packet in Figure 21 shall not contain the start of the payload from the next datagram.

10.12.4.3 Detection of last fragment of last UDP Packet on CICAM side (informative)

The last fragment of the last UDP packet can be detected either by checking for a TS packet containing an adaptation field or by implementing a timeout. The adaptation field method is usable only when the packet payload does not fit exactly into an integral number of TS packets.

11 Usage Rules Information version 3

CI Plus V1.3 [3], in clause 5.7.1, contains the definitions of CI Plus URI Versions 1 and 2. The present document specifies an extension to the CI Plus Usage Rules Information (URI) by defining CI Plus 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 Plus 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 V1.3 specification [3], Table 76 specifies the default values for CI Plus URI Version 3, including the new field *trick_mode_control_info*.

Table 90: Default values for CI Plus URI version 3

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	0b00000000
trick_mode_control_info	0b0
reserved bits	0b0

Table 91 defines the syntax of CI Plus URI version 3, which extends the existing URI versions 1 and 2 message syntax contained in clause 5.7.5.2 of CI Plus V1.3 [3].

Table 91: URI Version 3 message syntax

Field	Number of bits	Mnemonic
uri_message() {		
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) {		
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 Plus Browser (CI Plus Engine profile, application domain "CIEngineProfile1"), specified in CI Plus V1.3 [3], clause 12.2.
- Clause 12.3 specifies extensions to the CI application life cycle specified in CI Plus 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 Plus V1.3 [3], clause 12.6 and clause 6.5 of ETSI 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 Plus 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 Plus Engine Profile is extended to include the requirements for ICStreamingExtension including clause 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 Plus 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 Plus 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 Plus V1.3 [3] video scaling as a feature is not required in the CI Plus Engine Profile. The present document removes this exception, thus video scaling is a required feature in the CI Plus 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 Plus 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 Plus 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 realize this extension, the application domain identifier launch options in Table 12.8 in CI Plus 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 co-exist 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 ETSI TS 101 699 [2], clause 6.5.3 are extended and clarified in Table 96.

Table 96: AckCode values

AckCode	Meaning
0x00	Reserved for future use.
0x01	OK If option ADQ is missing, then the application execution environment shall attempt to load and execute the initial object specified in the *requestStart()* 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.
0x02	Wrong API Application domain not supported.
0x03	API busy Application domain supported but not currently available.
0x04	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.
0x05	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.
0x06 to 0x7F	Reserved for future use.
0x80 to 0xFF	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 Plus 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 Plus 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 Plus 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 Plus 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 ETSI 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 ETSI 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 ETSI 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 ETSI TS 102 809 [7] and ETSI 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 ETSI 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 Plus 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:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:mhp="urn:dvb:mhp:2009"
xmlns:dvb="urn:dvb:ciplus:xmlait:2013" targetNamespace="urn:dvb:ciplus:xmlait:2013">
    <xsd:import namespace="urn:dvb:mhp:2009" schemaLocation="mis_xmlait.xsd"/>
    <xsd:complexType name="CITransportType">
        <xsd:complexContent>
         <xsd:extension base="mhp:TransportProtocolDescriptorType">
             <xsd:sequence>
                <xsd:element name="initialObject" type="xsd:anyURI"
                    minOccurs="1" maxOccurs="1" />
             </xsd:sequence>
         </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd: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 ETSI 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 *enter_menu()* with a *requestStart()* APDU then the application identified in the *requestStart()* 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 Plus 1.3 [3] or ETSI 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.

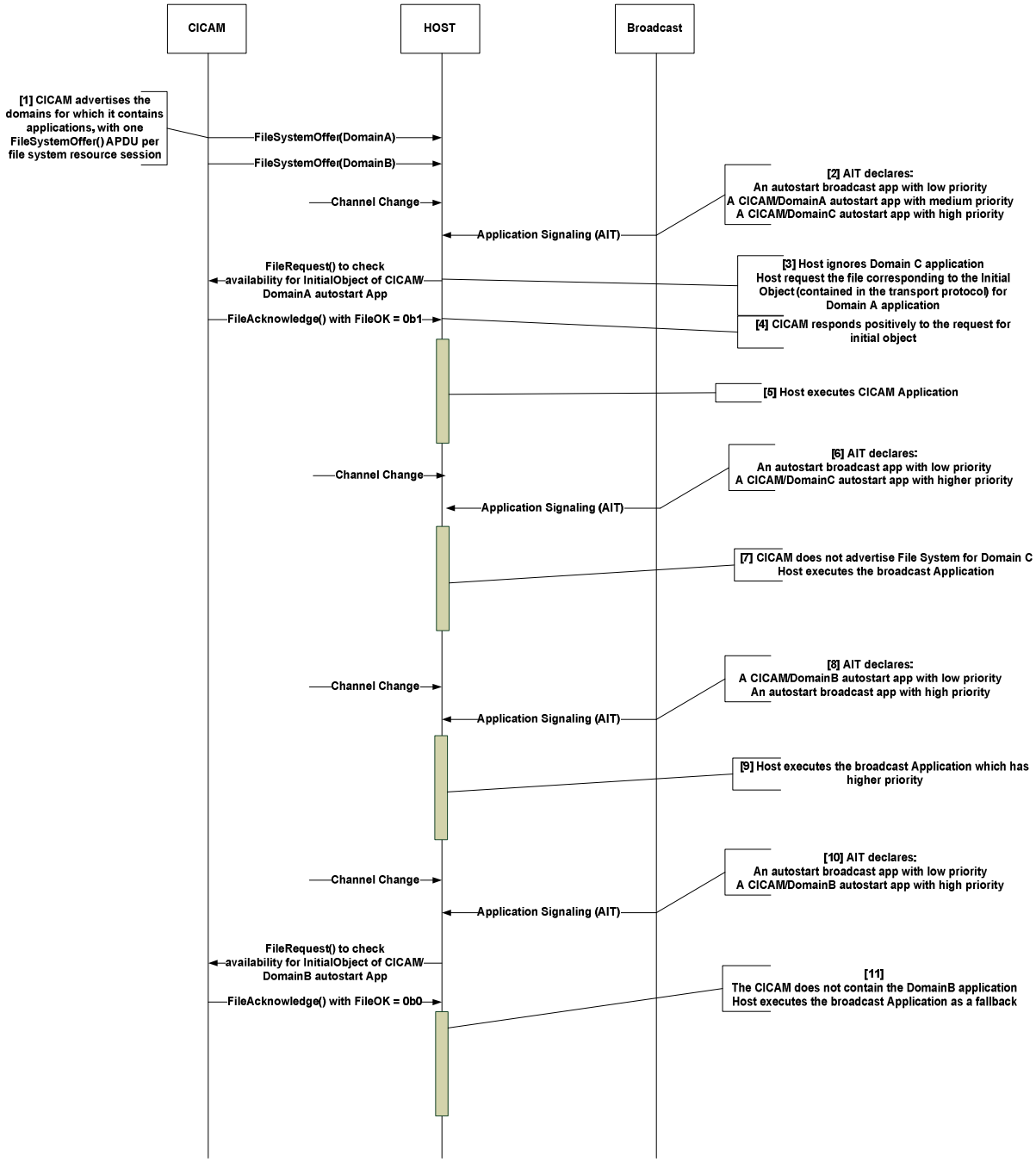

Figure 22: Broadcast signalled application launch example

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 F*ileSystemOffer()* 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.

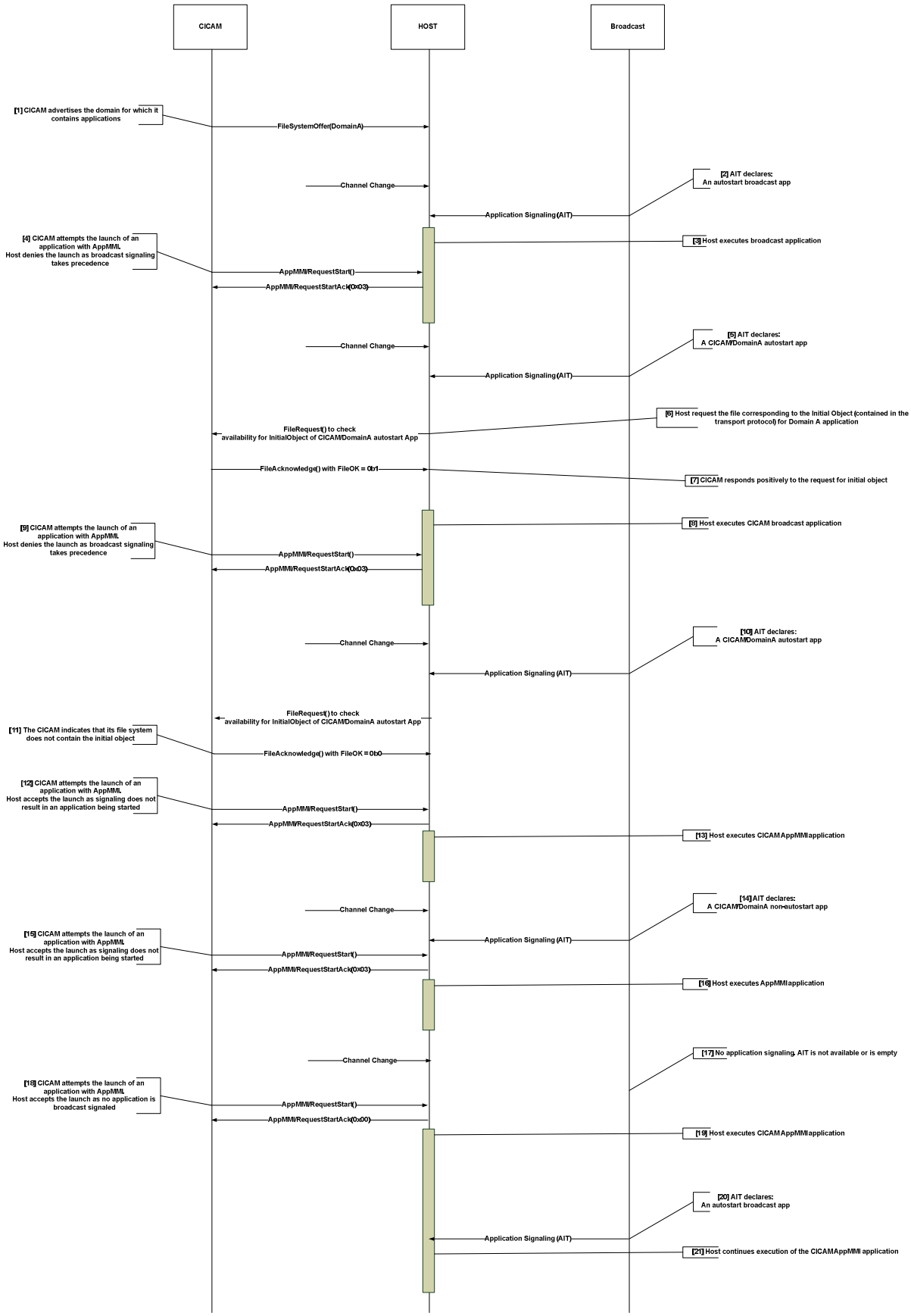

Figure 23: CICAM AppMMI application launch example

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.

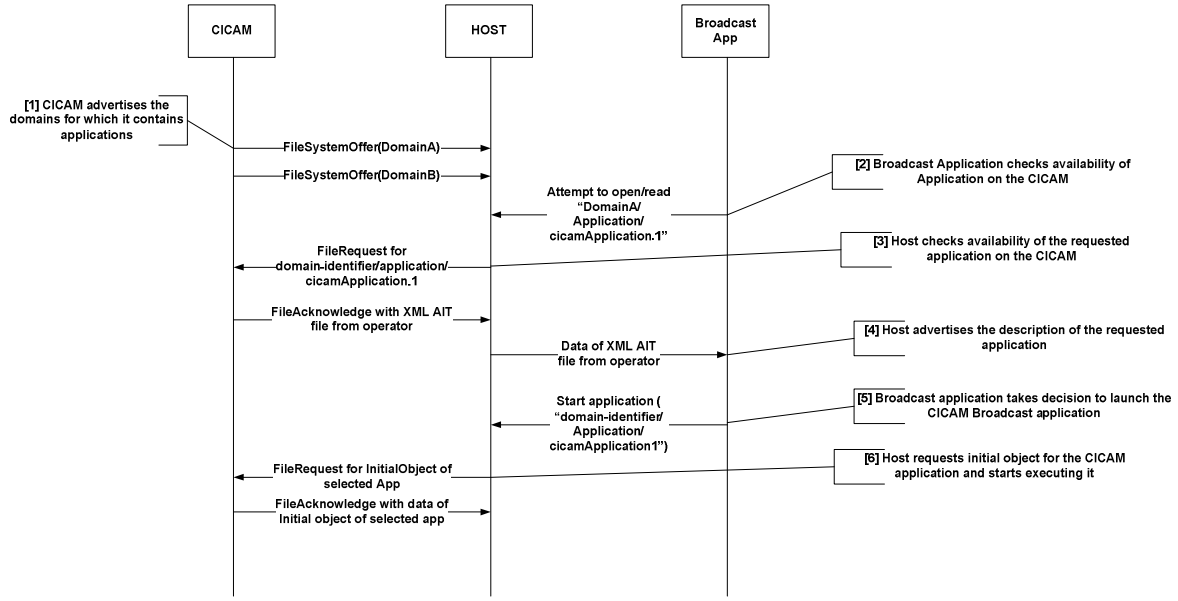

Figure 24: Broadcast application launches CICAM broadcast application example

1) The CICAM advertises the application domain "DomainA" for which it provides applications by use of the Auxiliary File System resource (*FileSystemOffer()* APDU).

2) 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.

3) 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 "AppDomainIdentfier" (ETSI 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 Plus browser, the value of the "AppDomainIdentfier" and "DomainIdentfier" is "CIMHEGP1".

The object which allows a Host to launch an application is referred to as the "InitialObject" (ETSI 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:

- From the sequence of data_broadcast_id listed in the application_capability_byte field of the *operator_search_start()* APDU in the operator profile resource.
- 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 IP-delivered 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 Plus 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_req APDU

Syntax	Number of bits	Mnemonic
tune_broadcast_req() {		
tune_broadcast_req_tag	24	uimsbf
length_field()		
reserved	5	uimsbf
tune_quietly_flag	1	uimsbf
keep_app_running_flag	1	uimsbf
pmt_flag	1	uimsbf
service_id	16	uimsbf
reserved	4	uimsbf
descriptor_loop_length	12	uimsbf
for (i=0; i<n; i++){		
descriptor()		
}		
if (pmt_flag == 1) {		
program_map_section()		
}		
}		

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 Plus V1.3 [3] for definitions of the other parameters in the *tune_broadcast_req()* 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
tune_triplet_req () {		
tune_lcn_triplet_tag	24	uimsbf
length_field()		
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
delivery_system_descriptor_tag	8	uimsbf
if (delivery_system_descriptor_tag == 0x7f){		
descriptor_tag_extension	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_lcn_req 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
tune_ip_req () {		
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_req*() 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
tuner_status_req() {		
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
tuner_status_reply() {		
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++){		
reserved	7	uimsbf
connected_flag	1	uimsbf
delivery_system_descriptor_tag	8	uimsbf
if (delivery_system_descriptor_tag == 0x7f){		
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 0b1indicates 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.

14.2.2 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 CENELEC EN 50221 [1], clause 8.3.1.

LTS_id: Local TS identifier.

15 Operator Profile version 2

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 Plus 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 Initialization 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 Plus 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 initialize 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 currently selected 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 preceeded by the CI Plus private data specifier descriptor. 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.

Table 104: CICAM Virtual Channel descriptor syntax

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++) {		
service_provider_name_char	8	uimsbf
}		
service_name_length	8	uimsbf
for (i=0; i<n; i++) {		
service_name_char	8	uimsbf
}		
event_information_length	8	uimsbf
for (i=0; i<n; i++) {		
event_information_char	8	uimsbf
}		
AppDomainIdentifierLength	8	uimsbf
for (i=0; i<n; i++) {		
AppDomainIdentifier	8	bslbf
}		
}		

The fields are defined as follows:

descriptor_tag: This 8-bit field shall be assigned the value 0xD2 and shall only be interpreted in the context of a CI Plus 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 ETSI 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 *operator_nit_management()* 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 "HLMMI". The application domain identifier "HLMMI" shall only be used in this descriptor.

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 *operator_nit_management()* APDU shall be sent by the Host at most once per CICAM NIT 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 *operator_nit()* 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
operator_nit_management() {		
operator_nit_management_tag	24	uimsbf
length_field()		
error_field	8	uimsbf
number_of_services	8	uimsbf
for (i=0; i<number_of_services; i++) {		
channel_issue_type	2	uimsbf
logical_channel_number	14	uimsbf
service_name_length	8	uimsbf
for (i=0; i<service_name_length; i++){		
char	8	uimsbf
}		
if (channel_issue_type == 0) {		
free_lcn_number	8	uimsbf
for (i=0; i<free_lcn_number; i++) {		
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.

Table 106: Error field values

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. 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.
0x04-0xFF	Reserved for future use.

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. If the number of services to be enumerated is greater than 255, the Host shall only enumerate the first 255 services which have not been stored. In this case, the CICAM shall not make any assumptions on whether any further services could not be stored.

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_descriptor* 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.

Table 107: Channel issue type values

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.

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 Plus 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_descriptor*s 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_v1.2.1.xsd in archive ts_103205v010401p0.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 8.8.9).
- 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 *uri_linkage_descriptor* is "cicam:", the Host shall use the *operator_osdt_request()* APDU to request the OSDT from the CICAM. In this case, the *min_polling_interval* field in the *uri_linkage_descriptor* 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 *min_polling_interval* 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

operator_osdt_request() {
  operator_osdt_request_tag
  length_field()
}
	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
for (i=0; i< osdt_file_data_length; i++) { 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 CENELEC 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.

- The *close_mmi()* APDU message may be sent by the CICAM to allow the user or an application to exit any dialogue in high-level MMI mode. The Host shall send this message when it decides to stop displaying a high-level MMI dialogue.

17 Operator Profile version 3

17.1 General

Operator Profile resource version 3 (with resource ID 0x008F1003) is an extension to version 3 and adds fine grained negotiation of the Host supported component types and the component types that are actually used in a service.

17.2 Operator Profile version 3 APDUs

17.2.1 operator_codec_verify_req APDU

The CICAM may send the *operator_codec_verify_req()* APDU to the Host to verify that the Host is capable of presenting a service with a given combination of components of different types. The Host shall reply with an *operator_codec_verify_reply()* APDU.

The CICAM should send the *operator_codec_verify_req()* APDU when the support for a specific service_type cannot be determined without additional information. ETSI EN 300 468 [10] Annex I defines the services_types which need additional information on their components. The combination of component_types shall describe completely the service.

NOTE: This may require the CICAM to insert additional component type if component descriptors are not present for all components in the network signalling.

Table 111: operator_codec_verify_req APDU

Syntax	Number of bits	Mnemonic
operator_codec_verify_req() {		
operator_codec_verify_req_tag	24	uimsbf
length_field()		
for (i=0; i<N; i++) {		
combination_tag	8	uimsbf
number_of_component_types	8	uimsbf
for (j=0; j<N; j++) {		
stream_content_ext	4	uimsbf
stream_content	4	uimsbf
component_type	8	uimsbf
}		
}		
}		

The fields are defined as follows:

operator_codec_verify_req_tag: The tag shall be set to value 0x9F9C10.

combination_tag: This 8-bit field specifies the tag allocated by the CICAM to the set of component types for which verification of support is requested.

number_of_component_types: This 8-bit field specifies the number of component types that are part of the combination for which support is requested.

stream_content_ext: This 4-bit field in combination with the stream_content field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

stream_content: This 4-bit field in combination with the stream_content_ext field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

component_type: This 8-bit field specifies the type of the component supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

17.2.2 operator_codec_verify_reply APDU

The Host shall send this APDU to the CICAM in response to the *operator_codec_verify_req()* APDU, to inform about the support of each of the combination requested for verification. This information is used by the CICAM to determine if installation of services corresponding to a combination is required.

The CICAM may send the *operator_codec_verify_reply()* APDU to the Host to verify that the Host is capable of presenting a service containing a given set of components of different types. The Host shall reply with an *operator_codec_verify_reply()* APDU.

The Host shall always include the combination tag in the reply for a combination that is supported.

Table 112: operator_codec_verify_reply APDU

Syntax	Number of bits	Mnemonic
operator_codec_verify_reply() {		
operator_codec_verify_reply_tag	24	uimsbf
length_field()		
for (i=0; i<N; i++) {		
combination_tag	8	uimsbf
}		
}		

The fields are defined as follows:

operator_codec_verify_reply_tag: The tag shall be set to value 0x9F9C11.

combination_tag: This 8-bit field specifies the tag allocated by the CICAM to the set of component types for which verification of support is requested. Only supported combination_tag shall be present in this list.

17.3 ciplus_advanced_service_descriptor

The *ciplus_advanced_service_descriptor* extends the *ciplus_service_descriptor* by a loop of component types. For services where the *service_type* alone does not fully allow a receiver to determine the specific type of the service (e.g. for HEVC service_types 0x1F or 0x20), this loop shall be used to indicate the specific type of the service.

A CICAM NIT may contain a mix of *ciplus_service_descriptors* and *ciplus_advanced_service_descriptors*. A CICAM NIT shall not contain a *ciplus_service_descriptors* and *a ciplus_advanced_service_descriptor* describing the same service with the same service_id.

Table 113: ciplus_ advanced_service_descriptor syntax

Syntax	No. of bits	Mnemonic
ciplus_ advanced_service_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
service_id	16	uimsbf
service_type	8	uimsbf
visible_service_flag	1	bslbf
selectable_service_flag	1	bslbf
logical_channel_number	14	uimsbf
service_provider_name_length	8	uimsbf
for (i=0; i<service_provider_name_length; i++) {		
char	8	uimsbf
}		
service_name_length	8	uimsbf
for (i=0; i<service_name_length; i++) {		
char	8	uimsbf
}		
number_of_component_types	8	uimsbf
for (i=0; i<number_of_component_types; i++) {		
stream_content_ext	4	uimsbf
stream_content	4	uimsbf
component_type	8	uimsbf
}		
}		

The additional fields (compared to CI Plus [3], clause N.1.2.3) are defined as follows:

number_of_component_types: This 8-bit field specifies the number of component types of this service.

stream_content_ext: This 4-bit field in combination with the stream_content field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

stream_content: This 4-bit field in combination with the stream_content_ext field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

component_type: This 8-bit field specifies the type of the component supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

18 CICAM Player version 2

18.1 General

CICAM Player resource version 2 (with resource ID 0x00930042) is an extension to version 1 and adds fine grained advertisement of the Host supported component types.

18.2 CICAM Player version 3 APDUs

18.2.1 CICAM_player_codec_verify_req APDU

The CICAM may send the *CICAM_player_codec_verify_req()* APDU to the Host to verify that the Host is capable of presenting a service with a given combination of components of different types. The Host shall reply with an *CICAM_player_codec_verify_reply()* APDU.

The combination of component_types shall describe completely the SPTS that the CICAM will output to the Host.

Table 114: CICAM_player_codec_verify_req APDU

Syntax	Number of bits	Mnemonic
CICAM_player_codec_verify_req() {		
CICAM_player_codec_verify_req_tag	24	uimsbf
length_field()		
for (i=0; i<N; i++) {		
combination_tag	8	uimsbf
number_of_component_types	8	uimsbf
for (j=0; j<N; j++) {		
stream_content_ext	4	uimsbf
stream_content	4	uimsbf
component_type	8	uimsbf
}		
}		
}		

The fields are defined as follows:

CICAM_player_codec_verify_req_tag: The tag shall be set to value 0x9FA010

combination_tag: This 8-bit field specifies the tag allocated by the CICAM to the set of component types for which verification of support is requested.

number_of_component_types: This 8-bit field specifies the number of component types that are part of the combination for which support is requested.

stream_content_ext: This 4-bit field in combination with the stream_content field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

stream_content: This 4-bit field in combination with the stream_content_ext field specifies the type of stream supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

component_type: This 8-bit field specifies the type of the component supported by the Host. The coding of this field is the same as that used within the Component descriptor, specified in clause 6.2.8 of ETSI EN 300 468 [10].

18.2.2 CICAM_player_codec_verify_reply APDU

The Host shall send this APDU to the CICAM in response to the *CICAM_player_codec_verify_req()* APDU, to inform about the support of each of the combination requested for verification.

The CICAM may send the *CICAM_player_codec_verify_reply()* APDU to the Host to verify that the Host is capable of presenting a service containing a given set of components of different types. The Host shall reply with an *CICAM_player_codec_verify_reply()* APDU.

Table 115: CICAM_player_codec_verify_reply APDU

Syntax	Number of bits	Mnemonic
CICAM_player_codec_verify_reply() {		
CICAM_player_codec_verify_req_tag	24	uimsbf
length_field()		
for (i=0; i<N; i++) {		
combination_tag	8	uimsbf
}		
}		

The fields are defined as follows:

CICAM_player_codec_verify_req_tag: The tag shall be set to value 0x9FA011.

combination_tag: This 8-bit field specifies the tag allocated by the CICAM to the set of component types for which verification of support is requested. Only supported *combination_tag* shall be present in this list.

19 Usage Rules Information (URI) version 5

Versions 1 and 2 of this message are defined in clause 5.7.1 of CI Plus 1.3 [3]. Version 3 of this message is defined in clause 11. Version 4 is defined in clause 22.2.1. The present document specifies an extension to the CI Plus Usage Rules Information (URI) by defining version 5 of the CI Plus URI for the purpose of defining additional rules for the usage of ECP content.

Table 116: URI version 5 message syntax

Syntax	No. of bits	Mnemonic
uri_message() {		
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) {		
rct_copy_control_info	1	uimsbf
}		
else {		
reserved = 0	1	bslbf
}		
reserved_future_use	1	bslbf
if (emi_copy_control_info == 11		
|| emi_copy_control_info == 10) {		
dot_copy_control_info	1	uimsbf
else {		
reserved = 0	1	bslbf
}		
if (emi_copy_control_info == 11) {		
rl_copy_control_info	10	uimsbf
}		
else {		
reserved = 0	10	bslbf
}		
if (emi_copy_control_info == 10) {		
trick_mode_control_info	1	uimsbf
}		
else {		
reserved = 0	1	bslbf
}		
if (emi_copy_control_info == 11) {		
ecp_control_info	3	uimsbf
}		
else {		
reserved = 0	3	bslbf
}		
reserved_future_use	34	bslbf
}		

protocol_version: This 8-bit field indicates the version of the URI definition. It shall be set to 0x05. A device made according to this version of the CI Plus specification shall understand values 0x01 to 0x05, and ignore URI messages that have a protocol_version value that it does not support.

aps_copy_control_info: This 2-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

emi_copy_control_info: This 2-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

ict_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3]. The Host shall use the ICT bit to control a constrained image quality on both analog and digital outputs.

rct_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

dot_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

ecp_control_info: This parameter describes the Enhanced Content Protection (ECP) bits which define (i) the setting of content protection on outputs and (ii) the authorization for content retention as explained in Table 117:

Table 117: Allowed Values for ecp_control_info

Value in Binary	Comment
000	Content is not ECP protected
001	Reserved
010	Content is ECP protected Export to HDCP-protected outputs is allowed Export to other protected outputs is allowed
011	Reserved
100	Content is ECP protected Only allowed export is to output protected by either HDCP 1.4 [27] or HDCP 2.2 [24]
101	Content is ECP protected Only allowed export is to output protected by either HDCP 1.4 [27] or HDCP 2.2 [24] No retention is allowed. The rl_copy_control_info field is ignored
110	Content is ECP protected Only allowed export is to output protected by HDCP 2.2 according to HDCP 2.2 [24]
111	Content is ECP protected Only allowed export is to output protected by HDCP 2.2 according to HDCP 2.2 [24] No retention is allowed. The rl_copy_control_info field is ignored

In the style of clause 5.7.4 of the CI Plus V1.3 specification [3], Table 118 specifies the default values for CI Plus URI Version 5, including the new field *ecp_control_info*.

Table 118: Default values for CI Plus URI version 5

Field	Default Initial Value
protocol version	0x05
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	0b0000000000
trick_mode_control_info	0b0
ecp_control_info	0b000
reserved bits	0b0

20 Clarifications on usage of Content Control System ID

Clause 11.3.1.2 of CI Plus 1.3 [3] defines how a Host can advertise the set of CC system IDs that it supports by use of the *cc_system_id_bitmask* as indicated in the *cc_open_cnf()* APDU.

From the list of the CC system IDs that are supported by the Host, the CICAM shall select the applicable CC system ID and use a *cc_system_id_bitmask* with only one bit set, corresponding to the identification of the selected CC system ID in subsequent *cc_data_req()* APDU (as defined in clause 11.3.1.3 of CI Plus 1.3 [3]), *cc_sac_data_req()* APDU (as defined in clause 11.3.1.7 of CI Plus 1.3 [3]) and *cc_sac_data_cnf()* APDU (as defined in clause 11.3.1.7 of CI Plus 1.3 [3]) sent to the Host. The CICAM determines the Root of Trust (set of certificates) to be used according to the selected CC System ID.

On reception of the first *cc_data_req()* from the CICAM (corresponding to the step 5 of the Authentication steps as described in clause 6.2.1 of CI Plus 1.3 [3]), the Host shall consider the selected CC System ID according to the only bit set on the *cc_system_id_bitmask*. The Host determines the Root of Trust (set of certificates) to be used according to the selected CC System ID.

Similarly, the Host shall use a *cc_system_id_bitmask* with only one bit set, corresponding to the identification of the selected CC system ID in subsequent *cc_data_cnf()* APDU (as defined in clause 11.3.1.3 of CI Plus 1.3 [3]) , *cc_sac_data_req()* APDU (as defined in clause 11.3.1.7 of CI Plus 1.3 [3]) and *cc_sac_data_cnf()* APDU (as defined in clause 11.3.1.7 of CI Plus 1.3 [3]) sent to the Host.

The *cc_system_id_bitmask* contained in the first *cc_data_req()* APDU from the CICAM shall be used by both the Host and the CICAM for all subsequent APDU exchanges on the current session until this session is closed.

The allocation of *cc_system_id_bitmask* bits (each representing one CC system ID) is specified in ETSI TS 101 162 [11]. *The bit value 0b1000 0000 is reserved for an extension mechanism which may be defined in a future version of the present document.*

21 Application Information version 5

21.1 General

Version 1 of this resource with the resource type 0x00020041 is defined in clause 8.4.2 of CENELEC EN 50221 [1].

Version 2 of this resource with the resource type 0x00020042 is defined in clause 5 of ETSI TS 101 699 [2].

Version 3 of this resource with the resource type 0x00020043 is defined in clause 11.1 of CI Plus [3].

Version 4 of this resource with the resource type 0x00020044 is defined in clause 14.2.

This clause defines version 5 of this resource with the resource type 0x00020045. Clause 21.2 APDU extensions defines new APDUs.

NOTE: This version of the Application Information resource was previously defined in DVB Bluebook A173--2 [i.2], which has been merged into the present document.

21.2 APDU extensions

21.2.0 Application Information resource summary

Table 119: Application Information resource summary

Resource					Application Object		Direction
Name	Resource Identifier	Class	Type	Version	APDU Tag	Tag value	Host ↔ CICAM
Application Information	00 02 00 45	2	1	5	application_info_enq	9F 80 20	➤
					application_info	9F 80 21	➤
					enter_menu	9F 80 22	➤
					request_CICAM_reset	9F 80 23	➤
					data_rate_info	9F 80 24	➤
					enter_cicam_channel	9F 80 25	➤
					hds_request	9F 80 26	➤
					hds_confirm	9F 80 27	➤
					power_down_notice	9F 80 28	➤
					power_down_ok	9F 80 29	➤

21.2.1 Host diagnostic screen APDUs

21.2.1.0 General

The Host Diagnostic Screen APDUs are used to request the Host to display a diagnostic screen about itself to the viewer. This information can be useful to the viewer when troubleshooting viewing of controlled content, for example by reading the information to a customer support agent on the phone. Since not all Hosts implement a diagnostic screen, or it may not be possible for the Host to display it at all times, a status confirmation is defined to inform the CICAM about the action performed.

21.2.1.1 HDS request APDU

Table 120: hds_request APDU syntax

Syntax	No. of bits	Mnemonic
`hds_request() {`		
`  hds_req_tag`	24	uimsbf
`  length_field()`	variable	
`  reserved_future_use`	7	bslbf
`  hds_command`	1	uimsbf
`}`		

hds_req_tag: This 24-bit field indicates the type of APDU message, and shall be coded according to Table 119.

hds_command: This 1-bit field indicates the requested action. It shall be coded according to Table 121.

Table 121: hds_command coding

hds_command	Description
0	Display a diagnostic screen
1	Stop displaying any diagnostic screen

21.2.1.2 HDS confirmation APDU

Table 122: hds_confirm APDU syntax

Syntax	No. of bits	Mnemonic
hds_confirm() {		
hds_cnf_tag	24	uimsbf
length_field()	variable	
reserved_future_use	6	bslbf
hds_status	2	uimsbf
}		

hds_cnf_tag: This 24-bit field indicates the type of APDU message, and shall be coded according to Table 119.

hds_status: This 2-bit field indicates the current state of the diagnostic screen. It shall be coded according to Table 123.

Table 123: hds_status coding

hds_status	Description
0	The Host is showing a diagnostic screen
1	The Host is not showing a diagnostic screen
2	The Host does not implement a diagnostic screen
3	reserved for future use

This means that after a successful *hds_request()* with *hds_command*=0, the Host shall send a *hds_confirm()* with *hds_status*=0, and that after a successful *hds_request()* with *hds_command*=1, the Host shall send a *hds_confirm()* with *hds_status*=1.

When the user terminates the display of the diagnostic screen, the Host shall send this APDU to the CICAM with the *hds_status* set to 1.

21.2.2 Power down APDUs

21.2.2.0 General

The *power_down_notice()* APDU is used to announce to the CICAM that the Host intends to remove power from the CICAM soon. The CICAM is given a grace period. If the CICAM does not wish to use the grace period, it can inform the Host by using the *power_down_ok()* APDU so that power can be removed immediately.

21.2.2.1 Power down notice APDU

This APDU shall be sent by the Host before entering a power mode where the CICAM will be powered down. Host shall give a minimum of 30 seconds notice before powering down the CICAM. This is allowing the CICAM to perform operations which should not be delayed like e.g. a firmware update or a channel map update.

During the 30 second period, the CICAM may request sessions with resources. Unless a session is open with a resource whose definition provides for preventing the Host from powering off the CICAM, the CICAM will unconditionally be powered off at the end of the 30 second period.

Since the Host may already be in the process of entering a standby mode when sending this APDU, the CICAM can not expect to be able to interact with the user after this message. Requests to open a session to any MMI resources may hence result in "resource busy" responses during the grace period.

The scheduled entitlement update already implies a 30 second period at the end of the operation (see clause 14.7.4.4.3 of CI Plus [3]). Hence, the CICAM shall not rely on receiving a power down notice APDU during that operation.

Since it may not always be possible for the Host to give the CICAM an early notice of being powered down (e.g. when mains power of the Host is unplugged), or there may be situations in which the Host powers down the CICAM without changing its own power mode (e.g. when resetting the CICAM), the CICAM shall expect to be powered down at any time without prior notice. These situations may also occur after the Host has sent this APDU, during the 30 second notice period.

Table 124: power_down_notice APDU syntax

Syntax	No. of bits	Mnemonic

power_down_notice() {
  power_down_notice_tag
  length_field()
}
	24
	variable
		uimsbf

power_down_notice_tag: This 24-bit field indicates the type of APDU message, and shall be coded according to Table 119.

21.2.2.2 Power down OK APDU

This APDU shall be sent by the CICAM after receiving a *power_down_notice()* APDU to signal the Host that CICAM does not have any more pending operation to perform and that the Host may power it down immediately. This message should be sent as early as possible to reduce power consumption of the system.

Table 125: power_down_ok APDU syntax

Syntax	No. of bits	Mnemonic

power_down_ok() {
  power_down_ok_tag
  length_field()
}
	24
	variable
		uimsbf

power_down_ok_tag: This 24-bit field indicates the type of APDU message, and shall be coded according to Table 119.

22 Content Control version 4 (multi-stream version 2)

22.1 General

Version 1 of this resource with the resource type 0x008C1001 is defined in clause 11.3 of CI Plus [3].

Version 2 of this resource with the resource type 0x008C1002 is defined in clause 11.3 of CI Plus [3].

Version 3 of this resource with the resource type 0x008C1003 is defined in clause 11.

Clause 6.4.3 of the present document further extends this resource by defining a new resource type of 0x008C1041 and 0x008C1003.

This clause defines version 2 of this resource with the resource type 0x008C1042 for multi-stream, and version 4 of this resource with the resource type 0x008C1004 for single stream. Clause 22.2 defines new messages for the SAC.

NOTE: This version of the Content Control resource was previously defined in DVB Bluebook A173-2 [i.2], which has been merged into the present document.

22.2 Protocol extensions

22.2.1 Usage Rules Information (URI) version 4

Versions 1 and 2 of this message are defined in clause 5.7.1 of CI Plus [3]. Version 3 of this message is defined in clause 11. The present clause specifies an extension to the CI Plus Usage Rules Information (URI) by defining version 4 of the CI Plus URI. Version 5 of this message is defined in clause 19.

Table 126: URI version 4 message syntax

Syntax	No. of bits	Mnemonic
uri_message() {		
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) {		
rct_copy_control_info	1	uimsbf
}		
else {		
reserved = 0	1	bslbf
}		
reserved_future_use	1	bslbf
if (emi_copy_control_info == 11		
|| emi_copy_control_info == 10) {		
dot_copy_control_info	1	uimsbf
else {		
reserved = 0	1	bslbf
}		
if (emi_copy_control_info == 11) {		
rl_copy_control_info	10	uimsbf
}		
else {		
reserved = 0	10	bslbf
}		
if (emi_copy_control_info == 10) {		
trick_mode_control_info	1	uimsbf
}		
else {		
reserved = 0	1	bslbf
}		
reserved_future_use	37	bslbf
}		

protocol_version: This 8-bit field indicates the version of the URI definition. It shall be set to 0x04. A device made according to this version of the CI Plus specification shall understand values 0x01 to 0x04, and ignore URI messages that have a *protocol_version* value that it does not support.

aps_copy_control_info: This 2-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

emi_copy_control_info: This 2-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

ict_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

rct_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

dot_copy_control_info: This 1-bit field shall be coded according to clause 5.7.5.3 of CI Plus [3].

rl_copy_control_info: This 10-bit field describes the retention limit of the recording and/or time-shift of content from the time that it is retained. It shall be coded according to Table 127.

Table 127: rl_copy_control_info coding

rl_copy_control_info	Description
0x000	90 minutes
0x001	6 hours
0x002	12 hours
0x003 to 0x3FE	1 to 1 020 units of 24 hours
0x3FF	unlimited

EXAMPLE: A rl_copy_control_info value of 0x3FE indicates a retention limit of 1 020 units of 24 hours, totalling 24 480 hours, or 2 years and 290 days.

trick_mode_control_info: This 1-bit field shall be coded according to clause 11.

22.2.2 Output Control protocol

Table 128 shows the Output Control protocol for use with version 2 of this resource with the resource type 0x008C1042 for multi-stream. The *output_num* parameter shall use a *datatype_id* set to value 51, and indicates the number of simultaneous outputs of different CI Plus controlled content received from a CICAM, which the Host is allowed to output to client devices via trusted protection systems.

Table 128: Output Control protocol

Step	Action	APDU	Content		
1	CICAM informs the Host about allowed number of outputs	cc_sac_data_req	send_datatype_nbr=1		
			index	**datatype_id**	**datatype_len**
			0	51 (output_num)	8 bits
			request_datatype_nbr=1		
			index	**datatype_id**	
			0	51 (output_num)	
2	Host sends acknowledgement to the CICAM	cc_sac_data_cnf	send_datatype_nbr=1		
			index	**datatype_id**	**datatype_len**
			0	51 (output_num)	8 bits

The CICAM shall wait for the acknowledgement from the Host before sending any further Output Control protocol message. The CICAM may only run this protocol at power-up after the URI version negotiation, and only again when the number of permitted outputs has actually changed, i.e. when the *output_num* parameter has a different value than in the last Output Control protocol message from the CICAM that the Host has acknowledged.

Upon receiving an Output Control protocol message from the CICAM, the Host shall apply the new limit as soon as possible. To acknowledge that the new limit is in effect, the Host shall send a confirmation Output Control protocol message with the *output_num* parameter set to the new limit value.

Until the CICAM runs this protocol for the first time, no restriction on the number of additional outputs is implied for the Host.

NOTE 1: This default behaviour is identical with prior versions of this resource, which did not impose any limit.

NOTE 2: To preserve the Output Control state per CICAM after the protocol has been run for the first time, the Host may need to persistently store it across power cycles, removal of the CICAM, and similar events for as long as recordings made with a particular CICAM are available.

The following rules shall apply:

a) Host internal rendering of a single CI Plus controlled content shall always be possible, and shall not count as an output. The output of the same CI Plus controlled content that the Host is internally rendering shall also not count as an output.

NOTE 3: On a Host device with no integrated display, as for example a set-top box, the content sent on the connection to the display device (for example via HDMI to a TV set), is considered as internal rendering.

b) Host internal rendering of additional CI Plus controlled content as picture-in-picture shall always be possible, and shall not count as an output.

c) Host internal and Host controlled recordings (according to URI settings) shall always be possible and shall not count as an output. Host internal rendering of a single CI Plus protected recording shall not count as an output.

d) Playback of CI Plus protected recordings without internal rendering shall count as an output.

The following examples illustrate some of the above rules.

EXAMPLE 1: A TV set is receiving CI Plus controlled broadcast services. It is also connected via the in-home network to a tablet device that implements a trusted protection system. If the *output_num* parameter is set to zero, the TV set may only output to the tablet the CI Plus controlled content it is displaying. If the *output_num* parameter is set to one, the TV set may output to the tablet a different CI Plus controlled service that is not currently displaying on the TV.

EXAMPLE 2: Extending from the previous example, if the *output_num* parameter is set to zero, but two tablet devices that implements a trusted protection system are connected to the TV set via the in-home network, the TV set may output to both tablets the CI Plus controlled content it is displaying.

23 Content Control version 5 (multi-stream version 3)

23.1 Negotiation of optional features

23.1.1 General

The Content Control (CC) single stream resource version 5 (type 0x008C1005) and multi-stream resource version 3 (type 0x008C1043) include a negotiation protocol to decouple optional features from the CC resource version. This enables the definition of optional features without the need for new CC resource versions, and provides a mechanism to ensure compatibility between devices with support for optional capabilities.

23.1.2 Negotiation protocol for optional features

Devices that support CC resource version 5 or multi-stream version 3 shall support the protocol for the negotiation of optional features for use with those resources.

The CICAM may only run the optional features protocol at power-up after the URI version negotiation. Before this protocol is completed, the Host and CICAM shall not send data defined for the optional features in Table 130, clause 24.1.

Table 129 shows the optional features negotiation protocol. Initially, the CICAM shall request the list of optional features supported by the Host (step 1). The Host shall respond to the CICAM with the list of optional features that it supports (step 2). The CICAM shall evaluate the features supported by the Host. When a feature is not supported by both devices, data defined for that feature shall not be exchanged, in particular the CICAM shall not intiate exchange of data defined for that feature, and the corresponding optional feature shall be disabled.

Table 129: Negotiation protocol for optional features

Step	Action	APDU	Content		
1	CICAM requests the bitmask of optional features supported by the Host	cc_sac_data_req	send_datatype_nbr = 0		
			request_datatype_nbr = 1		
			index	**datatype_id**	
			0	52 (optional_features)	
2	Host sends the bitmask of supported optional features	cc_sac_data_cnf	send_datatype_nbr = 1		
			index	**datatype_id**	**datatype_len**
			0	52 (optional_features)	32 bits

23.1.3 Data type for optional features

The *optional_features* parameter is a 32-bit bitmask used by the Host to convey the list of optional features it supports to the CICAM, and shall use a *datatype_id* value set to 52. Each bit corresponds to one single optional feature.

If an optional feature is supported by the Host, it shall set the corresponding bit (='1'), else the Host shall not set the corresponding bit (='0'). See Table 130 for the mapping between features and corresponding bits.

24 Optional Content Control Features

24.1 List of optional features

Table 130 contains the list of optional CC features defined in the present document. Additional optional features may be added in future versions of the present document.

NOTE: Bit 0 refers to the least significant bit of the *optional_features* parameter.

Table 130: CC resource optional features

Bit	Feature	Definition
0	Overt Watermarking	Clause 24.2
1-31	Reserved	

24.2 Overt Watermarking

24.2.1 General

The overt watermarking feature enables the CICAM to request the Host to display a text string over the video as a means of identifying the source of the content (e.g. a subscriber ID). Overt watermarking is sometimes referred to as fingerprinting, although the latter term has an additional meaning relating to the identification of video content that is outside the scope of the present document.

The CICAM shall only send data related to the overt watermarking feature after verifying the Host supports overt watermarking using the negotiation protocol for optional features.

24.2.2 Overt watermark content

The overt watermark shall consist of a 7-bit prinTable ASCII text string, with configurable font size, colour, opacity and position.

24.2.3 Logical plane, rendering and displaying

The Host uses overt watermark text and parameters communicated by the CICAM to render the watermark, i.e. to construct the graphical element representing the watermark, which the Host then displays over its video plane.

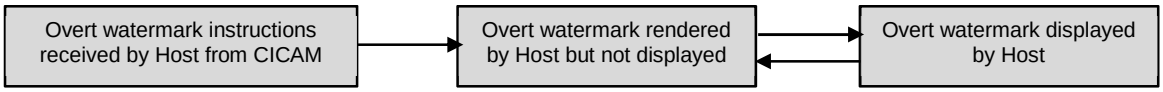

Figure 25: Overt watermark rendering and display sequence (informative)

Overt watermarking uses a logical plane of 1 280 pixels horizontally by 720 pixels vertically (16:9 ratio) as its coordinate system with (0,0) at the top left corner.

The Host shall display the overt watermarking logical plane on top of its fullscreen video plane. In this case, the logical plane shall be proportionally scaled by the Host to the same size as its fullscreen video plane.

When the Host's video plane is not displayed full-screen (e.g. downscaled and re-positioned within an interactive application), the Host may:

- Re-position and proportionally downscale the watermark logical plane to the position and size of the video plane (the watermark is also downscaled). In this case, the watermark logical plane shall be positioned to fully overlap the video logical plane.
- Only proportionally re-position the watermark over the downscaled video plane (the watermark position is adjusted, but it is not downscaled).

The Host shall be capable of rendering and displaying both subtitles and an overt watermark simultaneously. When their positions overlap, subtitles shall be displayed on top of an overt watermark.

The Host is not required to ensure the overt watermark is visible if the portion of video over which the watermark is intended to be displayed is temporarily obscured by graphics or subtitles, moved off-screen, or the video is blanked. This may occur when the Host displays graphics such as interactive applications, High-Level MMI, EPG or settings, as well as in the context of screen blanking caused by cc_PIN_event() (as defined in CI Plus Specification V1.3 [3]).

If the watermark text would overflow beyond the boundary of the watermark logical plane, the Host may cease to display the watermark or clip the watermark so that only the portion within the boundary of the watermark logical plane is visible.

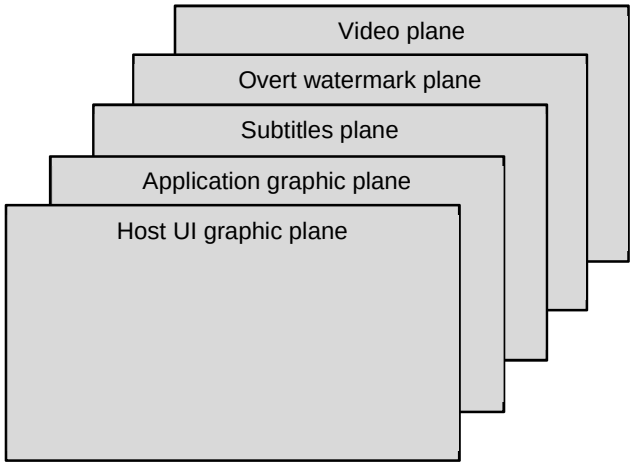

Figure 26: Logical plane ordering (informative)

No requirements are defined or implied for the number of hardware graphics planes supported by the Host.

The Host may render the overt watermark text using either bitmap or vector fonts. For the latter, anti-aliasing is optional.

24.2.4 Overt watermarking data types

24.2.4.1 wm_instructions (CICAM → Host)

The *wm_instructions* parameter of variable length contains the watermark text and rendering parameters, as defined in Table 131, and shall use a *datatype_id* value set to 53.

Table 131: wm_instructions data type

Syntax	Number of bits	Mnemonic
wm_instructions () {		
position_x	16	uimsbf
position_y	16	uimsbf
font_size	8	uimsbf
red	8	uimsbf
green	8	uimsbf
blue	8	uimsbf
alpha	8	uimsbf
text		uimsbf
}		

position_x, position_y: These 16-bit fields define the coordinates of the top-left corner of the watermark, within the overt watermark logical plane defined in clause 24.2.3.

font_size: This 8-bit field defines the size of the watermark text according to Table 132:

Table 132: font_size values

Size	Value	Percentage of logical plane height	Font height in logical pixel (informative)	Maximum text length (informative)
Smaller	0x00	3%	22	80
Small	0x01	4%	28	60
Normal	0x02	5%	36	50
Large	0x03	7%	50	35
Larger	0x04	10%	72	25
Reserved	0x05-0xFF			

red, green, blue: These 8-bit RGB components define the watermark text colour.

alpha: This 8-bit field defines the watermark opacity. From 0x00 indicating fully transparent text, to 0xFF indicating fully opaque text.

text: This field contains the watermark text, restricted to 7-bit ASCII prinTable characters (code points 0x20 to 0x7E, as defined in ETSI EN 300 468 [10], Figure A.1), up to a maximum length of 80 characters.

NOTE: Line feed, carriage return, tabulation and all other control characters are prohibited.

The text length restriction is determined based upon the longest displayable text string when using the smallest allowed font size ('smaller' 0x00) and placing the watermark at a leftmost position within the watermark logical plane. When the watermark is placed further to the right, and/or a larger font size is used, the Host may not be able to fully display the watermark (see Table 132 for maximum text length based on *font_size*).

The Host is not required to adjust the position, font size or text content to prevent clipping.

24.2.4.2 wm_rendered (Host → CICAM)

The 8-bit *wm_rendered* parameter indicates whether the Host is rendering the watermark or not, as defined in Table 133, and shall use a *datatype_id* value set to 54.

Table 133: wm_rendered data type values

Value	Watermark status
0x00	Not rendered
0x01	Rendered
0x02-0xFF	Reserved

When the value of *wm_rendered* is set to 0x00 'not rendered', this may be due to a lack of Host resources, the Host may not be in the correct operating mode, or the *LTS_id* and/or *program_number* may not match any of the services currently displayed by the Host.

When the value of wm_rendered is set to 0x01 'rendered', the Host shall render and display the watermark. The Host shall always display the rendered watermark, except as allowed in clause 24.2.3. In such cases, the Host is not required to inform the CICAM, the status remains 'rendered'.

24.2.4.3 wm_off (CICAM → Host)

The *wm_off* parameter contains no data and shall use a *datatype_id* value set to 55. It instructs the Host to stop rendering the watermark.

24.2.5 Overt watermarking protocol

24.2.5.1 Displaying an overt watermark

Table 134 describes the overt watermarking protocol used by a CICAM to request the Host to display an overt watermark.

In single-stream mode, the *LTS_id* shall be set to the fixed value of 0x47, but it may be safely ignored by the device receiving the message. In multi-stream mode, the *LTS_id* identifies the Local TS that the watermark protocol instructions apply to.

Table 134: Overt watermarking protocol for displaying an overt watermark

Step	Action	APDU	Content		
1	CICAM sends the watermark instructions to the Host	cc_sac_data_req	send_datatype_nbr = 3		
			index	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits
			2	53 (wm_instructions)	variable length
			request_datatype_nbr = 3		
			index	**datatype_id**	
			0	50 (LTS_id)	
			1	26 (program_number)	
			2	54 (wm_rendered)	
2	Host reports the status of the watermark rendering	cc_sac_data_cnf	send_datatype_nbr = 3		
			index	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits
			2	54 (wm_rendered)	8 bits

A CICAM shall only ask the Host to display an overt watermark for a service it can descramble after it has received the CA_PMT for that service from the Host during programme selection, with *ca_pmt_cmd_id* set to *ok_descrambling*.

To request the display of an overt watermark (step 1), a CICAM shall send instructions to the Host (*wm_instructions* as defined in clause 24.2.4.1) containing the watermark text and rendering parameters, as well as the *LTS_id* and *program_number* of the service over the video of which the Host is to display the watermark.

The Host shall always respond to a CICAM's request by reporting the watermark rendering status to the CICAM (*wm_rendered* as defined in clause 24.2.4.2), with the associated *LTS_id* and *program_number* (step 2). Failure to respond to the CICAM within 5 seconds constitutes a failure of the Content Control system.

A CICAM shall wait for a response from the Host before sending any further Overt Watermarking protocol message.

A CICAM shall not send watermark instructions to the Host more frequently than once every 5 seconds.

When the Host receives watermark instructions from a CICAM, if the CICAM was selected to descramble that service and *LTS_id* and *program_number* match those of a currently displayed scrambled service, the Host shall render and display the watermark over the video of that service within 5 seconds, except as allowed in clause 24.2.3, and the Host shall report a rendering status of 'rendered' (0x01).

If the Host does not render the watermark when requested by a CICAM, it shall report a rendering status of 'not rendered' (0x00). When the Host does not render the requested watermark and the service is not being recorded in either "Timeshift" or "Unattended_Recording" operating mode (see clause 24.2.6), the CICAM may choose to stop descrambling the service until the watermark is rendered by the Host. The CICAM shall provide user feedback when descrambling is stopped, for example via an MMI message.

The Host shall not render watermark instructions with an associated *LTS_id* and *program_number* combination that either do not match those of a scrambled service that is currently displayed or being recorded, or that match a service in the clear. In either of these cases, the Host shall report a rendering status of 'not rendered' (0x00).

24.2.5.2 Host status updates

Table 135 describes the overt watermarking protocol used by the Host to provide overt watermark rendering status updates to the CICAM.

Table 135: Overt watermarking protocol for providing status updates

Step	Action	APDU	Content		
1	Host reports a change in the status of the watermark rendering	cc_sac_data_req	send_datatype_nbr = 3		
			index	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits
			2	54 (wm_rendered)	8 bits
			request_datatype_nbr = 2		
			index	**datatype_id**	
			0	50 (LTS_id)	
			1	26 (program_number)	
2	CICAM acknowledges receipt of new watermark rendering status	cc_sac_data_cnf	send_datatype_nbr = 2		
			index	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits

The Host shall inform the CICAM of updates to the watermark rendering status (step 1). The CICAM shall acknowledge receipt of rendering status updates (step 2).

24.2.5.3 CICAM removing an overt watermark

Table 136 describes the overt watermarking protocol used by the CICAM to request the Host to remove an overt watermark.

Table 136: Overt watermarking protocol for removing an overt watermark

Step	Action	APDU	Content		
1	CICAM instructs the Host to stop rendering the watermark	cc_sac_data_req	send_datatype_nbr = 2		
			index	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits
			2	55 (wm_off)	0 bits
			request_datatype_nbr = 3		
			index	**datatype_id**	
			0	50 (LTS_id)	
			1	26 (program_number)	
			2	54 (wm_rendered)	
2		cc_sac_data_cnf	send_datatype_nbr = 3		

Step	Action	APDU	Content		
	Host reports the status of the watermark rendering		**index**	**datatype_id**	**datatype_len**
			0	50 (LTS_id)	8 bits
			1	26 (program_number)	16 bits
			2	54 (wm_rendered)	8 bits

The CICAM may request the Host to stop rendering the watermark for a given combination of *LTS_id* and *program_number* at any time (step 1). When receiving such a request (*wm_off* as defined in clause 24.2.4.3), the Host shall stop rendering the watermark as soon as possible, and shall acknowledge the CICAM's request by reporting the updated watermark rendering status to the CICAM, with the associated *LTS_id* and *program_number* combination (step 2).

24.2.5.4 Host removing an overt watermark

The Host shall stop rendering the watermark when a channel change occurs that causes the Host to no longer display the watermark's corresponding programme. In such a case, the Host shall not report a change in watermark rendering status, the CA_PMT sent during programme selection with a different *program_number* and a *ca_pmt_list_management* set to 'first' or 'only' shall be sufficient to inform the CICAM of the channel change and consequent watermark removal.

24.2.6 Overt watermarking during recording and playback

When recording a CA protected service, the Host shall associate overt watermark instructions (i.e. text, parameters, rendering status) received with the recording, at the time they were received, so they can be displayed at the same position in the content during playback.

When the Host has initiated a recording using the Record Start Protocol (defined in clause 11.3.4.4 of the CI Plus Specification V1.3 [3]) and is in "Timeshift" or "Unattended_Recording" operating mode, the Host shall report a rendering status of 'not rendered' when reporting the overt watermark rendering status to the CICAM. During a recording in "Timeshift" or "Unattended_Recording" operating mode, the CICAM shall not stop descrambling due to the Host reporting a rendering status of 'not rendered'.

During a recording in "Watch_and_Buffer" operating mode, the CICAM and Host shall use the same overt watermarking behaviour as when descrambling and displaying live content respectively (as defined in clause 24.2.5), however the Host shall still associate overt watermark instructions received with the recording.

During playback of recorded content, the Host shall render and display overt watermarks associated with the recording, except as allowed in clause 24.2.3, at their position in the recording. The Host shall not report the rendering status to the CICAM. If the Host is not able to render the overt watermark, it shall not display the portion of recorded content that required a watermark to be added.

24.2.7 Overt watermarking during playback of IP-delivered content

24.2.7.1 General

The same overt watermarking protocol shall be used during playback of IP-delivered content as when descrambling and displaying live broadcast content, as defined in clause 24.2.5, with the following differences.

24.2.7.2 Overt watermarking in CICAM player mode

When in CICAM player mode, a CICAM shall only ask the Host to display an overt watermark for content it can descramble after it has received *CICAM_player_start_reply()* from the Host during establishment of the CICAM player session, with *input_status* set to 0x00 (OK).

The Host shall stop rendering the watermark after sending *CICAM_player_stop()* to the CICAM or after receiving either *CICAM_player_end()* or *CICAM_player_status_error()* from the CICAM, all of which signal playback of the watermark's corresponding content has ended. In such cases, the Host is not required to report a change in watermark rendering status.

24.2.7.3 Overt watermarking in Host player mode

When in Host player mode, a CICAM shall only ask the Host to display an overt watermark for content it can descramble after it has sent *sd_start_reply()* to the Host with *transmission_status* set to 0x00 (ready to receive) and *drm_status* set to 0x00 (decryption possible), during establishment of the Host player session.

The Host shall stop rendering the watermark after it has finished sending media samples to the CICAM and has sent *ca_pmt()* to the CICAM, which signals playback of the watermark's corresponding content has ended. The Host may also stop rendering the watermark when the CICAM reports a *drm_status* other than 0x00 (decryption possible) for a Sample Track containing video content, using *sd_update_reply()*. In such cases, the Host is not required to report a change in watermark rendering status.

24.2.8 Overt watermarking sequence (informative)

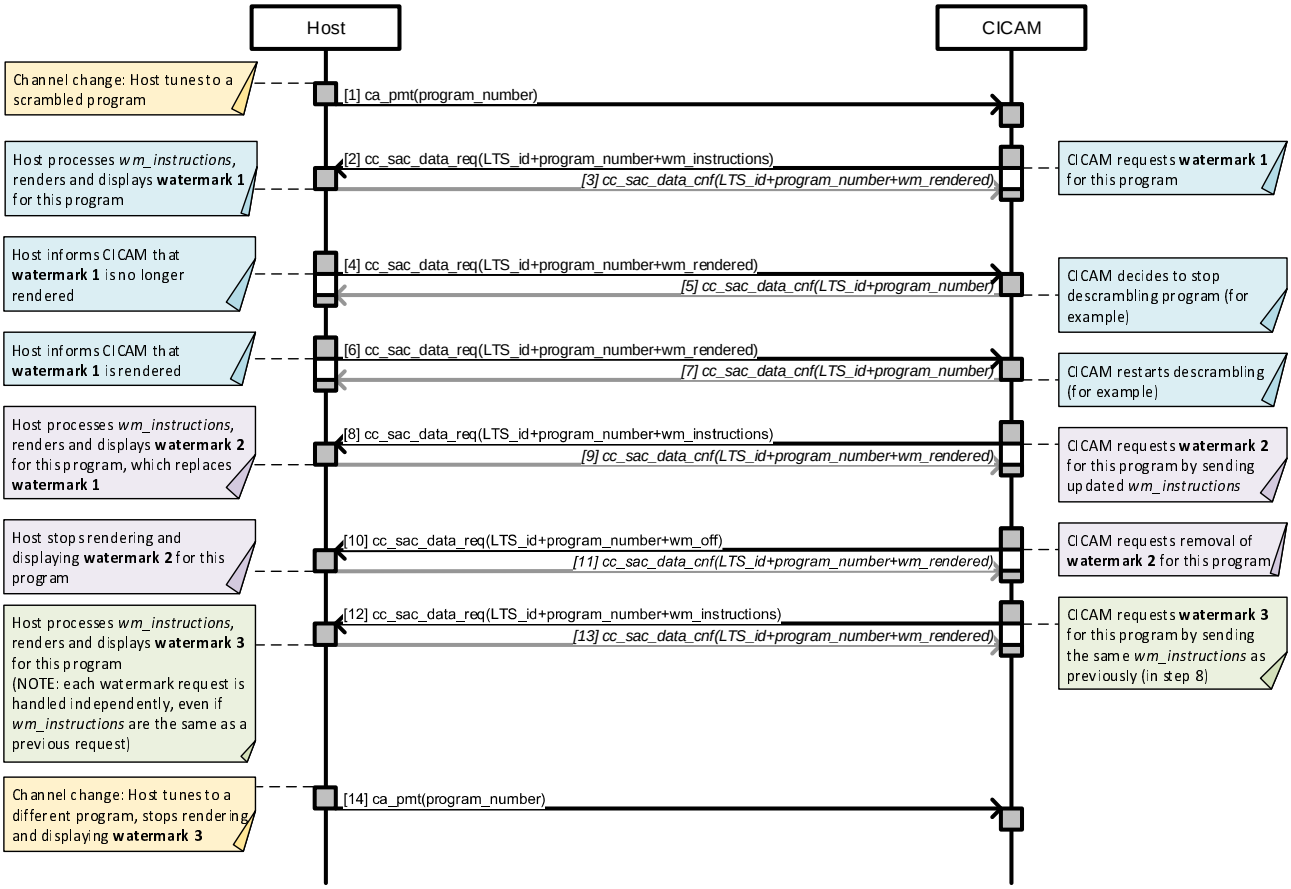

Figure 27

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

Resource					Application Object		Direction		
Name	**Resource Identifier**	**Class**	**Type**	**Vers.**	**APDU Tag**	**Tag value**	**Host**	**CICAM**	**Spec**
Resource Manager	00 01 00 41	1	1	1	profile_enq	9F 80 10	←→		CENELEC EN 50221 [1]
					profile	9F 80 11	←→		
					profile_change	9F 80 12	←→		
	00 01 00 42	1	1	2	profile_enq	9F 80 10	←→		ETSI TS 101 699 [2]
					profile	9F 80 11	←→		
					profile_change	9F 80 12	←→		
					CICAM_id_send	9F 80 13	←		
					CICAM_id_command	9F 80 14	→		
Application Information	00 02 00 41	2	1	1	application_info_enq	9F 80 20	→		CENELEC EN 50221 [1]
					application_info	9F 80 21	←		
					enter_menu	9F 80 22	→		
	00 02 00 42	2	1	2	application_info_enq	9F 80 20	→		ETSI TS 101 699 [2]
					application_info	9F 80 21	←		
					enter_menu	9F 80 22	→		
	00 02 00 43	2	1	3	application_info_enq	9F 80 20	→		CI Plus 1.3 [3]
					application_info	9F 80 21	←		
					enter_menu	9F 80 22	→		
					request_CICAM_reset	9F 80 23	←		
					data_rate_info	9F 80 24	→		
	00 02 00 44	2	1	4	application_info_enq	9F 80 20	→		Present document
					application_info	9F 80 21	←		
					enter_menu	9F 80 22	→		
					request_CICAM_reset	9F 80 23	←		
					data_rate_info	9F 80 24	→		
					enter_cicam_channel	9F 80 25	→		
	00 02 00 45	2	1	5	application_info_enq	9F 80 20	→		Present document
					application_info	9F 80 21	←		
					enter_menu	9F 80 22	→		
					request_CICAM_reset	9F 80 23	←		
					data_rate_info	9F 80 24	→		
					enter_cicam_channel	9F 80 25	→		
					hds_request	9F 80 26	←		
					hds_confirm	9F 80 27	→		
					power_down_notice	9F 80 28	→		
					power_down_ok	9F 80 29	←		
Conditional Access Support	00 03 00 41	3	1	1	ca_info_enq	9F 80 30	→		CENELEC EN 50221 [1]
					ca_info	9F 80 31	←		
					ca_pmt	9F 80 32	→		
					ca_pmt_reply	9F 80 33	←		
	00 03 00 81 (see note 1)	3	2	1	ca_info_enq	9F 80 30	→		Present document
					ca_info	9F 80 31	←		
					ca_pmt	9F 80 32	→		
					ca_pmt_reply	9F 80 33	←		
Host Control	00 20 00 41	32	1	1	tune	9F 84 00	←		CENELEC EN 50221 [1]
					replace	9F 84 01	←		
					clear_replace	9F 84 02	←		
					ask_release	9F 84 03	→		
	00 20 00 42	32	1	2	tune	9F 84 00	←		CI Plus 1.3 [3]
					replace	9F 84 01	←		
					clear_replace	9F 84 02	←		
					ask_release	9F 84 03	→		
					tune_broadcast_req	9F 84 04	←		
					tune_reply	9F 84 05	→		
					ask_release_reply	9F 84 06	←		
	00 20 00 43	32	1	3	tune	9F 84 00	←		Present document
					replace	9F 84 01	←		
					clear_replace	9F 84 02	←		
					ask_release	9F 84 03	→		
					tune_broadcast_req	9F 84 04	←		
					tune_reply	9F 84 05	→		
					ask_release_reply	9F 84 06	←		
					tune_lcn_req	9F 84 07	←		
					tune_ip_req	9F 84 08	←		
					tune_triplet_req	9F 84 09	←		
					tune_status_req	9F 84 0A	←		

Resource					Application Object		Direction		
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host	CICAM	Spec
					tune_status_reply	9F 84 0B	→		
	00 20 00 81 (see note 1)	32	2	1	ask_release	9F 84 03	→		Present document
					tune_broadcast_req	9F 84 04	←		
					tune_reply	9F 84 05	→		
					ask_release_reply	9F 84 06	←		
					tune_lcn_req	9F 84 07	←		
					tune_ip_req	9F 84 08	←		
					tune_triplet_req	9F 84 09	←		
					tune_status_req	9F 84 0A	←		
					tune_status_reply	9F 84 0B	→		
Date-Time	00 24 00 41	36	1	1	date_time_enq	9F 84 40	←		CENELEC EN 50221 [1]
					date_time	9F 84 41	→		
MMI	00 40 00 41	64	1	1	close_mmi	9F 88 00	→		CENELEC EN 50221 [1]
					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	←		
					keypress	9F 88 06	→		
					enq	9F 88 07	←		
					answ	9F 88 08	→		
					menu_last	9F 88 09	←		
					menu_more	9F 88 0A	←		
					menu_answ	9F 88 0B	→		
					list_last	9F 88 0C	←		
					list_more	9F 88 0D	←		
					subtitle_segment_last	9F 88 0E	←		
					subtitle_segment_more	9F 88 0F	→		
					display_message	9F 88 10	←		
					scene_end_mark	9F 88 11	←		
					scene_done	9F 88 12	←		
					scene_control	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	←		
	00 40 00 81 (see note 1)	64	2	1	close_mmi	9F 88 00	→		Present document
					display_control	9F 88 01	←		
					display_reply	9F 88 02	→		
					enq	9F 88 07	←		
					answ	9F 88 08	→		
					menu_last	9F 88 09	←		
					menu_more	9F 88 0A	←		
					menu_answ	9F 88 0B	→		
					list_last	9F 88 0C	←		
					list_more	9F 88 0D	←		
Low Speed Comms.	00 60 xx x1	96		1	comms_cmd	9F 8C 00	←		CENELEC EN 50221 [1]
					connection_descriptor	9F 8C 01	←		
					comms_reply	9F 8C 02	→		
					comms_send_last	9F 8C 03	←		
					comms_send_more	9F 8C 04	←		
					comms_rcv_last	9F 8C 05	→		
					comms_rcv_more	9F 8C 06	→		
	00 60 xx x2	96		2	comms_cmd	9F 8C 00	←		CI Plus 1.3 [3]
					connection_descriptor	9F 8C 01	←		
					comms_reply	9F 8C 02	→		
					comms_send_last	9F 8C 03	←		
					comms_send_more	9F 8C 04	←		
					comms_rcv_last	9F 8C 05	→		
					comms_rcv_more	9F 8C 06	→		
	00 60 xx x3	96		3	comms_cmd	9F 8C 00	←		CI Plus 1.3 [3]
					connection_descriptor	9F 8C 01	←		
					comms_reply	9F 8C 02	→		
					comms_send_last	9F 8C 03	←		
					comms_send_more	9F 8C 04	←		
					comms_rcv_last	9F 8C 05	→		
					comms_rcv_more	9F 8C 06	→		
	00 60 xx x4	96		4	comms_cmd	9F 8C 00	←		Present document
					connection_descriptor	9F 8C 01	←		
					comms_reply	9F 8C 02	→		
					comms_send_last	9F 8C 03	←		
					comms_send_more	9F 8C 04	←		
					comms_rcv_last	9F 8C 05	→		
					comms_rcv_more	9F 8C 06	→		
					comms_info_req	9F 8C 07	←		
					comms_info_reply	9F 8C 08	→		
					comms_IP_config_req	9F 8C 09	←		

Resource					Application Object		Direction		
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host	CICAM	Spec
					comms_IP_config_reply	9F 8C 0A	→		
Content Control	00 8C 10 01	140	64	1	cc_open_req	9F 90 01	←		CI Plus 1.3 [3]
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req	9F 90 07	←		
					cc_sac_data_cnf	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
	00 8C 10 02	140	64	2	cc_open_req	9F 90 01	←		CI Plus 1.3 [3]
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req	9F 90 07	←		
					cc_sac_data_cnf	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 03	140	64	3	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req	9F 90 07	←		
					cc_sac_data_cnf	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 04	140	64	4	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req	9F 90 07	←		
					cc_sac_data_cnf	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 05	140	64	5	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req	9F 90 07	←		
					cc_sac_data_cnf	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		

Resource					Application Object		Direction		
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host	CICAM	Spec
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 41 (see note 1)	140	65	1	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req (see note 2)	9F 90 07	←		
					cc_sac_data_cnf (see note 2)	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 42 (see note 1)	140	65	2	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req (see note 2)	9F 90 07	←		
					cc_sac_data_cnf (see note 2)	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
	00 8C 10 43 (see note 1)	140	65	3	cc_open_req	9F 90 01	←		Present document
					cc_open_cnf	9F 90 02	→		
					cc_data_req	9F 90 03	←		
					cc_data_cnf	9F 90 04	→		
					cc_sync_req	9F 90 05	←		
					cc_sync_cnf	9F 90 06	→		
					cc_sac_data_req (see note 2)	9F 90 07	←		
					cc_sac_data_cnf (see note 2)	9F 90 08	→		
					cc_sac_sync_req	9F 90 09	←		
					cc_sac_sync_cnf	9F 90 10	→		
					cc_PIN_capabilities_req	9F 90 11	→		
					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	←		
					cc_PIN_playback	9F 90 16	→		
					cc_PIN_MMI_req	9F 90 17	→		
Host Language & Country	00 8D 10 01	141	64	1	Host_country_enq	9F 81 00	←		CI Plus 1.3 [3]
					Host_country	9F 81 01	→		
					Host_language_enq	9F 81 10	←		
					Host_language	9F 81 11	→		
CICAM_Upgrade	00 8E 10 01	142	64	1	cicam_firmware_upgrade	9F 9D 01	←		CI Plus 1.3 [3]
					cicam_firmware_upgrade_reply	9F 9D 02	→		
					cicam_firmware_upgrade_progress	9F 9D 03	←		
					cicam_firmware_upgrade_complete	9F 9D 04	←		
Operator Profile	00 8F 10 01	143	64	1	operator_status_req	9F 9C 00	→		CI Plus 1.3 [3]
					operator_status	9F 9C 01	←		
					operator_nit_req	9F 9C 02	→		
					operator_nit	9F 9C 03	←		
					operator_info_req	9F 9C 04	→		
					operator_info	9F 9C 05	←		
					operator_search_start	9F 9C 06	→		
					operator_search_status	9F 9C 07	←		
					operator_exit	9F 9C 08	→		
					operator_tune	9F 9C 09	←		
					operator_tune_status	9F 9C 0A	→		
					operator_entitlement_ack	9F 9C 0B	→		
					operator_search_cancel	9F 9C 0C	→		
	00 8F 10 02	143	64	2	operator_status_req	9F 9C 00	→		Present document
					operator_status	9F 9C 01	←		
					operator_nit_req	9F 9C 02	→		

Resource					Application Object		Direction		
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host	CICAM	Spec
					operator_nit	9F 9C 03	←		
					operator_info_req	9F 9C 04	→		
					operator_info	9F 9C 05	←		
					operator_search_start	9F 9C 06	→		
					operator_search_status	9F 9C 07	←		
					operator_exit	9F 9C 08	→		
					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	→		
					operator_osdt_reply	9F 9C 0E	←		
					operator_nit_management	9F 9C 0F	→		
	00 8F 10 03	143	64	3	operator_status_req	9F 9C 00	→		Present document
					operator_status	9F 9C 01	←		
					operator_nit_req	9F 9C 02	→		
					operator_nit	9F 9C 03	←		
					operator_info_req	9F 9C 04	→		
					operator_info	9F 9C 05	←		
					operator_search_start	9F 9C 06	→		
					operator_search_status	9F 9C 07	←		
					operator_exit	9F 9C 08	→		
					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	→		
					operator_osdt_reply	9F 9C 0E	←		
					operator_nit_management	9F 9C 0F	→		
					operator_codec_verify_req	9F 9C 10	←		
					operator_codec_verify_reply	9F 9C 11	→		
SAS	00 96 10 01	150	64	1	SAS_connect_rqst	9F 9A 00	→		CI Plus 1.3 [3]
					SAS_connect_cnf	9F 9A 01	←		
					SAS_data_rqst (see note 3)	9F 9A 02	←→		
					SAS_data_av (see note 3)	9F 9A 03	←→		
					SAS_data_cnf (see note 3)	9F 9A 04	←→		
					SAS_server_query (see note 3)	9F 9A 05	←→		
					SAS_server_reply (see note 3)	9F 9A 06	←→		
					SAS_async_msg	9F 9A 07	←→		
Application MMI	00 41 00 41	65	1	1	RequestStart	9F 80 00	←		ETSI TS 101 699 [2]
					RequestStartAck	9F 80 01	→		
					FileRequest	9F 80 02	→		
					FileAcknowledge	9F 80 03	←		
					AppAbortRequest	9F 80 04	←→		
					AppAbortAck	9F 80 05	←→		
	00 41 00 42	65	1	2	RequestStart	9F 80 00	←		CI Plus 1.3 [3]
					RequestStartAck	9F 80 01	→		
					FileRequest	9F 80 02	→		
					FileAcknowledge	9F 80 03	←		
					AppAbortRequest	9F 80 04	←→		
					AppAbortAck	9F 80 05	←→		
	00 41 00 43	65	1	3	RequestStart	9F 80 00	←		Present document
					RequestStartAck	9F 80 01	→		
					FileRequest	9F 80 02	→		
					FileAcknowledge	9F 80 03	←		
					AppAbortRequest	9F 80 04	←→		
					AppAbortAck	9F 80 05	←→		
	0 41 00 81 (see note 1)	65	2	1	RequestStart	9F 80 00	←		Present document
					RequestStartAck	9F 80 01	→		
					FileRequest	9F 80 02	→		
					FileAcknowledge	9F 80 03	←		
					AppAbortRequest	9F 80 04	←→		
					AppAbortAck	9F 80 05	←→		
Multi-stream	00 90 00 41 (see note 1)	144	1	1	CICAM_multistream_capability	9F 92 00	←		Present document
					PID_select_req	9F 92 01	←		
					PID_select_reply	9F 92 02	→		
Auxiliary File System	00 91 00 41	145	1	1	FileSystemOffer	9F 94 00	←		Present document
					FileSystemAck	9F 94 01	→		
					FileRequest	9F 94 02	→		
					FileAcknowledge	9F 94 03	←		
Sample Decryption	00 92 00 41	146	1	1	sd_info_req	9F 98 00	→		Present document
					sd_info_reply	9F 98 01	←		
					sd_start	9F 98 02	→		
					sd_start_reply	9F 98 03	←		
					sd_update	9F 98 04	→		
					sd_update_reply	9F 98 05	←		

Resource					Application Object		Direction		
Name	Resource Identifier	Class	Type	Vers.	APDU Tag	Tag value	Host	CICAM	Spec
CICAM Player	00 93 00 41	147	1	1	CICAM_player_verify_req	9F A0 00	→		Present document
					CICAM_player_verify_reply	9F A0 01	←		
					CICAM_player_capabilities_req	9F A0 02	←		
					CICAM_player_capabilities_reply	9F A0 03	→		
					CICAM_player_start_req	9F A0 04	←		
					CICAM_player_start_reply	9F A0 05	→		
					CICAM_player_play_req	9F A0 06	→		
					CICAM_player_status_error	9FA0 07	←		
					CICAM_player_control_req	9F A0 08	→		
					CICAM_player_info_req	9F A0 09	→		
					CICAM_player_info_reply	9F A0 0A	←		
					CICAM_player_stop	9F A0 0B	→		
					CICAM_player_end	9F A0 0C	←		
					CICAM_player_asset_end	9F A0 0D	←		
					CICAM_player_update_req	9F A0 OE	←		
					CICAM_player_update_reply	9F A0 0F	→		
	00 93 00 42	147	1	2	CICAM_player_verify_req	9F A0 00	→		Present document
					CICAM_player_verify_reply	9F A0 01	←		
					CICAM_player_capabilities_req	9F A0 02	←		
					CICAM_player_capabilities_reply	9F A0 03	→		
					CICAM_player_start_req	9F A0 04	←		
					CICAM_player_start_reply	9F A0 05	→		
					CICAM_player_play_req	9F A0 06	→		
					CICAM_player_status_error	9FA0 07	←		
					CICAM_player_control_req	9F A0 08	→		
					CICAM_player_info_req	9F A0 09	→		
					CICAM_player_info_reply	9F A0 0A	←		
					CICAM_player_stop	9F A0 0B	→		
					CICAM_player_end	9F A0 0C	←		
					CICAM_player_asset_end	9F A0 0D	←		
					CICAM_player_update_req	9F A0 OE	←		
					CICAM_player_update_reply	9F A0 0F	→		
					CICAM_player_codec_verify_req	9F A0 10	←		
					CICAM_player_codec_verify_reply	9F A0 11	→		

NOTE 1: This resource is applicable for multi-stream operation as specified in clause 6.
NOTE 2: The *cc_sac_data_req()* and *cc_sac_data_cnf()* APDUs use *LTS_id* as a datatype field for URI transmission and acknowledgment.
NOTE 3: The synchronous SAS APDUs are not used by the CI Plus Specification V1.3 [3] or the present document.

Annex B (informative): 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
Kp	256	CICAM's key precursor to Host for CCK	12
DHPH	2 048	DH Public Key Host	13
DHPM	2 048	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	2 048	The signature of Host DH public key	17
Signature_B	2 048	The signature of CICAM DH public key	18
auth_nonce	256	Random nonce of 256 bits generated by the CICAM and transmitted by the CICAM to the Host for use in the authentication protocol	19
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 by the Host. Format is 'uimsbf'	29
status_field	8	Status field in APDU confirm messages	30
Srm_data_hdcp	variable	SRM for HDCP (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 (Note 4)	43
Reserved	-	-	44
Reserved	-	-	45
Reserved	-	-	46

Key or variable	Size (bits)	Comments	Datatype id
Reserved	-	-	47
Reserved	-	-	48
csuv	8	Critical Security Update Version (Note 5)	49
LTS_id	8	Local TS Identifier	50
output_num	8	Number of additional, simultaneous outputs of CI Plus controlled content to client devices	51
optional_features	32	Bitmask signaling the optional Content Control features supported by the Host	52
wm_instructions	variable	Overt watermark text and rendering parameters	53
wm_rendered	8	Host overt watermark rendering status	54
wm_off	0	Instructs Host to disable overt watermark rendering	55
Reserved	-	-	56 to 255

NOTE 1: Certificate lengths are variable.
NOTE 2: SRMs for HDCP are defined in the HDCP specification [27]. First generation SRMs shall not exceed 5 kilobytes.
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.
NOTE 5: This definition replaces the reservation for SRM from CI Plus 1.3 [3] as defined in [i.3].

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 Plus 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 NIT1st loop
ciplus_service_descriptor	0xCC	CI Plus V1.3 [3] N.1.2.3	CICAM NIT2nd loop
hostname_descriptor	0xCD	CI Plus V1.3 [3] 14.2.1.2	LSC comms_cmd APDU
ci_protection_descriptor	0xCE	CI Plus V1.3 [3] 10.1.1.1	SDT
IP_descriptor	0xCF	CI Plus V1.3 [3] 14.2.1.1	LSC comms_cmd APDU
ciplus_initialization_vector_descriptor	0xD0	Present document, clause 7.5.5.4.2	SSP
ciplus_key_identifier_descriptor	0xD1	Present document, clause 7.5.5.4.3	SSP
cicam_virtual_channel_descriptor	0xD2	Present document, clause 15.3.1	CICAM NIT 1st loop
ciplus_advanced_service_descriptor	0xD3	Present document, clause 17.3	CICAM NIT 2nd loop
Reserved	0xD4 to 0xEF		
Host defined	0xF0 to 0xFE		SSP, SEP, FLT
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_v1.2.1.xsd" in archive ts_103205v010401p0.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:2015`.

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="osdt:SubRegionType" minOccurs="0"/>
   </sequence>
</complexType>

<complexType name="RegionListType">
   <sequence>
      <element name="PrimaryRegion" type="osdt:SubRegionType" minOccurs="0"/>
   </sequence>
   <attribute name="CountryCodes" type=" sds:ISO-3166-List"/>
</complexType>

<complexType name="TargetRegionType">
   <sequence>
      <element name="RegionList" type="osdt: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="osdt:ContentAttributesType" minOccurs="0"/>
      <choice>
         <element name="IPMulticastAddress" type="sds:McastType"/>
         <element name="RTSPURL" type="sds:RTSPURLType"/>
         <element name="UriBasedLocation" type="osdt:ExtendedURIType"/>
      </choice>
   </sequence>
   <attribute name="priority" type="integer" default="0"/>
</complexType>

<complexType name="ExtendedURIType">
   <sequence>
      <element name="URI" type="anyURI"/>
   </sequence>
   <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"/>
   <attribute 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="osdt:ServiceLocationType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="LCN" type="osdt:LCNType" minOccurs="0"/>
      <element name="TargetRegions" type="osdt:TargetRegionType" minOccurs="0"/>
      <element name="ServiceName" type="sds:MultilingualType" maxOccurs="unbounded"/>
      <element name="ApplicationLocation" type="anyURI" minOccurs="0"/>
      <element name="ServiceGenre" type="sds:Genre" minOccurs="0"/>
      <element name="ServiceType" type="sds:ServiceType" minOccurs="0"/>
```

```
      <element name="ContentAttributes" type="osdt:ContentAttributesType" minOccurs="0"/>
      <element name="BCG" type="sds:BCGOffering" minOccurs="0"/>
   </sequence>
</complexType>

<complexType name="IPServiceListType">
   <sequence>
     <element name="IPService" type="osdt:IPServiceType" maxOccurs="unbounded"/>
      <element name="BCG" type="sds:BCGOffering" minOccurs="0"/>
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
   <attribute name="Version" type="positiveInteger" use="required"/>
</complexType>
```

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	A type used to provide the details of the service.
UniqueIdentifier	The unique ID of the service. This ID should never be 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 (see note).
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 ETSI TS 102 809 [7].
ServiceGenre	The genre of the service.
ServiceType	The service type, as defined in ETSI 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.
NOTE: If this triplet matches the triplet of another service, it can be assumed that the services editorially carry the same content.	

D.1.3.3 IPServiceList and ServiceLocation

```
<element name="IPServiceList" type="osdt:IPServiceListType"/>
<element name="ServiceLocation" type="osdt: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"?>
<IPServiceList xmlns="urn:dvb:metadata:ciplus:osdt:2015"
   xmlns:tva="urn:tva:metadata:2012"
   xmlns:sds="urn:dvb:metadata:iptv:sdns:2008-1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:dvb:metadata:ciplus:osdt:2015  osdt.xsd" Version="1">
   <IPService>
```

```
      <UniqueIdentifier ServiceName="channel-1" DomainName="www.broadcaster.com"/>
      <ServiceLocation>
         <ContentAttributes>
            <AudioAttributes>
               <tva:Coding href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001#3.2">
                  <tva:Name>MPEG-1 Audio Layer II</tva:Name>
               </tva:Coding>
            </AudioAttributes>
            <VideoAttributes>
               <tva:Coding href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001#2.2.2">
                  <Name>MPEG-2 Video Main Profile @ Main Level</Name>
               </tva:Coding>
            </VideoAttributes>
         </ContentAttributes>
         <UriBasedLocation contentType="application/dash+xml">
            <URI>http://www.broadcaster.com/mpd/channel-1-sd.mpd</URI>
         </UriBasedLocation>
      </ServiceLocation>
      <ServiceLocation>
         <ContentAttributes>
            <AudioAttributes>
               <tva:Coding href="urn:dvb:metadata:cs:AudioCodecCS:2007#3.1">
                  <tva:Name>E-AC3</tva:Name>
               </tva:Coding>
            </AudioAttributes>
            <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>
            </VideoAttributes>
         </ContentAttributes>
         <UriBasedLocation contentType="application/dash+xml">
            <URI>http://www.broadcaster.com/mpd/channel-1-hd.mpd</URI>
         </UriBasedLocation>
      </ServiceLocation>
      <LCN LCN="1" selectable="true"/>
      <TargetRegions>
         <RegionList CountryCodes="GBR">
            <PrimaryRegion>
               <Region>London</Region>
            </PrimaryRegion>
         </RegionList>
      </TargetRegions>
      <ServiceName Language="eng">Channel 1</ServiceName>
   </IPService>
   <BCG DomainName="www.operator.com">
      <sds:BCG Id="BCG1">
         <sds:Name Language="eng">Operator BCG</sds:Name>
         <sds:TransportMode>
            <sds:HTTP Location="http://www.operator/com/BCG1/dvb/sdns"/>
         </sds:TransportMode>
      </sds:BCG>
   </BCG>
</IPServiceList>
```

Annex E (informative): Management of a DiSEqC™ 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 DiSEqC™ 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 DiSEqC™ switch, then the user has to set up the DiSEqC™ switch topology in the Host for the tune to succeed. This annex describes a simple discovery mechanism for DiSEqC™ switch topology. Here it is assumed that the LNB is of universal type and that the DiSEqC™ topology is a single DiSEqC™ 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 DiSEqC™, it will likely have a configuration menu with a Table associating DiSEqC™ positions with orbital positions (or at least satellite names). The discovery mechanism populates this Table - the "Orbital-to-DiSEqC™" 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 DiSEqC™ positions.

Figure E.1 shows the discovery procedure.

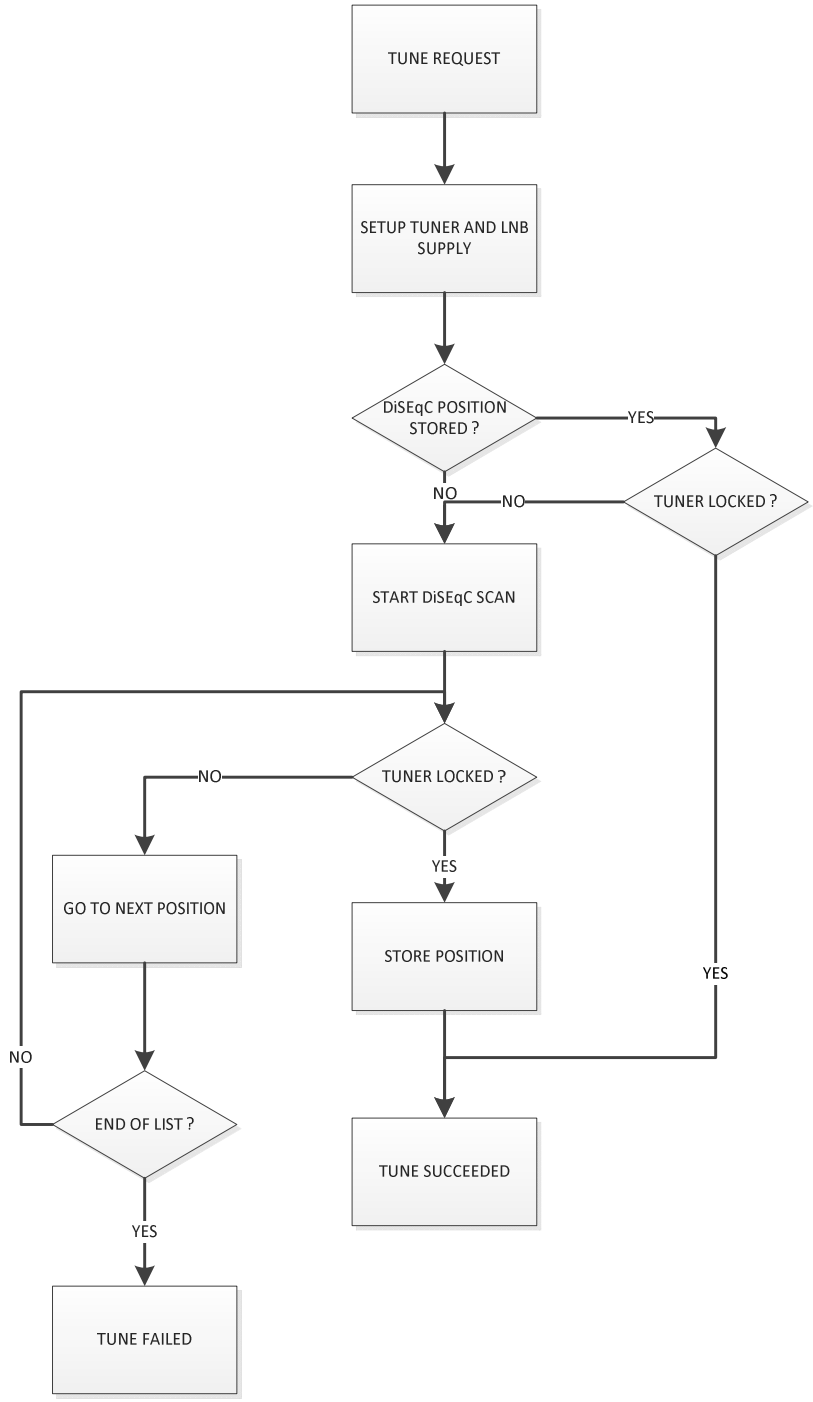

Figure E.1: CICAM-controlled DiSEqC™ switch installation procedure

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-DiSEqC™" map then the stored DiSEqC™ 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 DiSEqC™ 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 DiSEqC™ 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-DiSEqC™" 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 ETSI 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 Plus 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 *organization_id* and the *application_id*. 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 *organization_id* was 0x21and whose *application_id* 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.

Annex G (informative): Bibliography

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".

Annex H (informative): Change History

Date	Version	Information about changes
January 2019	1.4.1	Changes implemented: • Clarification of cc_system_id_bitmask bit (CC System ID) allocation and added a reference to a future RoT extension mechanism in clause 20. • Merged DVB Document A173-2 into clauses 21 and 22. • Clarification of Host behaviour related to power down notice in clause 21.2.2.1. • New Content Control v5 resource with a negotiation mechanism for optional CC resource features in clause 23 and 24.1. • New Overt Watermarking feature in clause 24.2 (optional CCv5 feature). • Editorial corrections.

History

Document history		
V1.1.1	March 2014	Publication
V1.2.1	November 2015	Publication
V1.3.1	December 2017	Publication
V1.4.1	May 2019	Publication