

**Superseded by a later version of this document.**

## **OpenCable™ Specifications**

### **CableCARD™ Interface 2.0 Specification**

**OC-SP-CCIF2.0-I02-050708**

**Issued**

#### **Notice**

This document is the result of a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the cable industry and its customers. This document may contain references to other documents not owned or controlled by CableLabs. Use and understanding of this document may require access to such other documents. Designing, manufacturing, distributing, using, selling, or servicing products, or providing services, based on this document may require intellectual property licenses for technology referenced in the document.

Neither CableLabs nor any member company is responsible to any party for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document, or any document referenced herein. This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, or fitness for a particular purpose of this document, or any document referenced herein.

© Copyright 2004-2005 Cable Television Laboratories, Inc.  
All rights reserved.

## Document Status Sheet

<b>Document Control Number:</b>	OC-SP-CCIF2.0-I02-050708			
<b>Document Title:</b>	CableCARD™ Interface 2.0 Specification			
<b>Revision History:</b>	D01 – August 31, 2004			
	D02 – December 30, 2004			
	I01 – March 31, 2005			
	I02 – July 8, 2005			
<b>Date:</b>	July 8, 2005			
<b>Responsible Editor</b>	Steve Young			
<b>Status:</b>	<del>Work in Progress</del>	<del>Draft</del>	Issued	<del>Closed</del>
<b>Distribution Restrictions:</b>	<del>Author Only</del>	<del>CL Member</del>	<del>CL Member/Vendor</del>	Public

### Key to Document Status Codes:

<b>Work in Progress</b>	An incomplete document, designed to guide discussion and generate feedback, which may include several alternative requirements for consideration.
<b>Draft</b>	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
<b>Issued</b>	A document that has undergone rigorous Member and vendor review, suitable for product design and development, cross-vendor interoperability, and for certification testing.
<b>Closed</b>	A stable document, reviewed, tested and validated, suitable to enable cross-vendor interoperability.

### TRADE MARKS:

DOCSIS®, eDOCSIS™, PacketCable™, CableHome®, CableOffice™, OpenCable™, CableCARD™, and CableLabs® are trademarks of Cable Television Laboratories, Inc.

# Contents

<b>1</b>	<b>SCOPE .....</b>	<b>1</b>
1.1	Introduction and Overview .....	1
1.2	Historical Perspective (Informative) .....	2
1.3	Requirements (Conformance Notation) .....	2
1.4	Numerical .....	3
<b>2</b>	<b>REFERENCES .....</b>	<b>4</b>
2.1	Normative References .....	4
2.2	Informative References .....	5
2.3	Reference Acquisition .....	5
<b>3</b>	<b>TERMS AND DEFINITIONS .....</b>	<b>7</b>
<b>4</b>	<b>ABBREVIATIONS AND ACRONYMS .....</b>	<b>11</b>
<b>5</b>	<b>MODEL OF OPERATION .....</b>	<b>14</b>
5.1	Advanced Cable Services .....	14
5.1.1	Interactive Program Guide (IPG) .....	14
5.1.2	Impulse Pay-Per-View (IPPV) .....	14
5.1.3	Video-on-Demand (VOD) .....	14
5.1.4	Interactive services .....	15
5.2	CableCARD Device Functional Description .....	15
5.2.1	Transport Stream Interface .....	15
5.2.2	Command Interface .....	16
5.3	Network Connectivity/OOB Signaling .....	16
5.4	Card Operational Modes .....	17
5.4.1	S-CARD in S-Mode .....	17
5.4.2	M-CARD in S-Mode .....	17
5.4.3	M-CARD in M-Mode .....	17
5.5	One-way Networks .....	17
5.6	Two-way Networks .....	18
5.7	Two-way Networks with DOCSIS .....	19
5.8	M-CARD Device Functional Description .....	20
5.9	Inband Interface - MPEG Data Flow .....	22
5.10	OOB Interface .....	23
5.10.1	QPSK .....	24
5.10.2	DSG .....	24
<b>6</b>	<b>DELETED .....</b>	<b>27</b>
<b>7</b>	<b>PHYSICAL INTERFACE .....</b>	<b>28</b>

<b>7.1 Electrical Characteristics .....</b>	<b>28</b>
<b>7.2 S-Mode Start-Up .....</b>	<b>30</b>
7.2.1 Card Port Custom Interface (0x341).....	30
<b>7.3 Interface Functional Description .....</b>	<b>30</b>
7.3.1 S-Mode.....	30
7.3.2 M-Mode .....	30
7.3.3 Card Signal Descriptions.....	30
7.3.4 Card Type Identification .....	34
7.3.5 Card Information Structure .....	36
7.3.6 MPEG Transport Interface.....	37
<b>7.4 Electrical Specifications .....</b>	<b>40</b>
7.4.1 DC Characteristics .....	40
7.4.2 AC Characteristics.....	45
<b>7.5 Mechanical Specifications .....</b>	<b>51</b>
7.5.1 Form Factor .....	51
7.5.2 Connector.....	51
7.5.3 Environmental .....	51
7.5.4 PC Card Guidance .....	51
7.5.5 Grounding/EMI Clips .....	51
7.5.6 Connector Reliability .....	51
7.5.7 Connector Durability .....	52
7.5.8 PC Card Environmental.....	52
<b>7.6 CPU Interface.....</b>	<b>52</b>
7.6.1 S-Mode.....	52
7.6.2 M-Mode .....	56
7.6.3 S-Mode Initialization and Operation .....	58
7.6.4 M-CARD Initialization and Operation.....	82
<b>8 COPY PROTECTION .....</b>	<b>85</b>
<b>9 COMMAND CHANNEL OPERATION .....</b>	<b>86</b>
<b>9.1 Session Layer .....</b>	<b>86</b>
9.1.1 S-Mode.....	86
9.1.2 M-Mode .....	86
9.1.3 Resources with Multiple Sessions.....	86
9.1.4 SPDU Structure .....	86
<b>9.2 Application Layer .....</b>	<b>90</b>
9.2.1 Resource Identifier Structure.....	90
<b>9.3 APDUs .....</b>	<b>91</b>
9.3.1 Interface Resource Loading .....	96
<b>9.4 Resource Manager .....</b>	<b>97</b>
9.4.1 profile_inq() .....	97
9.4.2 profile_reply() .....	98
9.4.3 profile_changed() .....	98
<b>9.5 Application Information.....</b>	<b>98</b>
9.5.1 application_info_req() .....	99

9.5.2	application_info_cnf()	101
9.5.3	server_query()	102
9.5.4	server_reply()	103
<b>9.6</b>	<b>Low Speed Communication</b>	<b>104</b>
<b>9.7</b>	<b>CA Support</b>	<b>105</b>
9.7.1	ca_info_inquiry	106
9.7.2	ca_info	106
9.7.3	ca_pmt	106
9.7.4	ca_pmt_reply	112
9.7.5	ca_update	115
<b>9.8</b>	<b>Host Control</b>	<b>117</b>
9.8.1	OOB_TX_tune_req	118
9.8.2	OOB_TX_tune_cnf	119
9.8.3	OOB_RX_tune_req	119
9.8.4	OOB_RX_tune_cnf	120
9.8.5	inband_tune_req	121
9.8.6	inband_tune_cnf	123
<b>9.9</b>	<b>Generic IPPV Support</b>	<b>123</b>
<b>9.10</b>	<b>System Time</b>	<b>123</b>
9.10.1	system_time_inq	124
9.10.2	system_time	124
<b>9.11</b>	<b>Man-Machine Interface (MMI)</b>	<b>125</b>
9.11.1	open_mmi_req	126
9.11.2	open_mmi_cnf	127
9.11.3	close_mmi_req	128
9.11.4	close_mmi_cnf	128
<b>9.12</b>	<b>M-Mode Device Capability Discovery</b>	<b>129</b>
9.12.1	stream_profile APDU	129
9.12.2	stream_profile_cnf APDU	130
9.12.3	program_profile APDU	130
9.12.4	programs_profile_cnf APDU	131
9.12.5	es_profile APDU	131
9.12.6	es_profile_cnf APDU	131
9.12.7	request_pids APDU	131
9.12.8	request_pids_cnf APDU	132
<b>9.13</b>	<b>Copy Protection</b>	<b>132</b>
<b>9.14</b>	<b>Extended Channel Support</b>	<b>133</b>
9.14.1	new_flow_req APDU	134
9.14.2	new_flow_cnf APDU	138
9.14.3	delete_flow_req APDU	140
9.14.4	delete_flow_cnf APDU	140
9.14.5	lost_flow_ind APDU	141
9.14.6	lost_flow_cnf APDU	141
9.14.7	DSG Mode	141
<b>9.15</b>	<b>Generic Feature Control</b>	<b>151</b>
9.15.1	Parameter Storage	152

9.15.2	Parameter Operation .....	152
9.15.3	Generic Feature Control Resource Identifier.....	154
9.15.4	Feature ID .....	155
9.15.5	Generic Feature Control APDUs .....	155
<b>9.16</b>	<b>Generic Diagnostic Support.....</b>	<b>164</b>
9.16.1	diagnostic_req APDU .....	165
9.16.2	diagnostic_cnf APDU .....	167
9.16.3	Diagnostic Report Definition .....	170
<b>9.17</b>	<b>Specific Application Support .....</b>	<b>180</b>
9.17.1	SAS_connect_rqst APDU.....	182
9.17.2	SAS_connect_cnf APDU .....	183
9.17.3	SAS_data_rqst APDU .....	183
9.17.4	SAS_data_av APDU.....	184
9.17.5	SAS_data_cnf APDU.....	184
9.17.6	SAS_server_query APDU .....	185
9.17.7	SAS_server_reply APDU.....	185
9.17.8	SAS Async APDU.....	185
<b>9.18</b>	<b>Card Firmware Upgrade.....</b>	<b>186</b>
9.18.1	Introduction.....	186
9.18.2	Implementation.....	188
9.18.3	Host Operation (Normative).....	188
9.18.4	Homing Resource.....	191
<b>9.19</b>	<b>Support for Common Download Specification.....</b>	<b>194</b>
9.19.1	Overview of Protocol .....	194
9.19.2	Operational Details.....	195
9.19.3	System Control Resource.....	201
<b>10</b>	<b>EXTENDED CHANNEL OPERATION.....</b>	<b>208</b>
10.1	Internet Protocol Flows .....	208
10.2	Flow Examples—QPSK Modem Case .....	208
10.3	Flow Examples— Embedded Cable Modem Case DSG Mode .....	209
10.4	Summary of Extended Channel Flow Requirement .....	211
10.5	System/Service Information Requirements .....	211
10.6	Link Layer .....	211
10.6.1	S-Mode.....	211
10.6.2	M-Mode .....	212
10.6.3	Maximum PDUs.....	212
10.7	Modem Models .....	212
10.7.1	Unidirectional Host Model .....	212
10.7.2	Bidirectional With Modem in Card .....	213
10.7.3	Bidirectional With Modem in Host .....	213
10.8	SI Requirements.....	213
10.9	EAS Requirements .....	213
10.10	XAIT Requirements .....	213

<b>ANNEX A</b>	<b>BASLINE HTML PROFILE SUPPORT .....</b>	<b>214</b>
<b>ANNEX B</b>	<b>ERROR HANDLING .....</b>	<b>221</b>
<b>ANNEX C</b>	<b>CRC-8 REFERENCE MODEL .....</b>	<b>234</b>
<b>ANNEX D</b>	<b>S-CARD ATTRIBUTE AND CONFIGURATION REGISTERS ....</b>	<b>235</b>
<b>ANNEX E</b>	<b>DOWNLOADINFOINDICATOR MESSAGE DETAIL FOR COMMON DOWNLOAD (NORMATIVE).....</b>	<b>242</b>
<b>APPENDIX I</b>	<b>REVISION HISTORY .....</b>	<b>246</b>

## Tables

Table 1.4–1 - Numerical Representation .....	3
Table 5.4–1 - Card/Host Combinations and Operating Modes .....	17
Table 5.8–1 - Card-Host Resource Communication .....	22
Table 5.8–2 - Resource Example Request .....	22
Table 7.1–1 - Card Interface Pin Assignments .....	28
Table 7.3–1 - Timing Relationship Limits .....	32
Table 7.3–2 - Transmission Signals.....	34
Table 7.3–3 - VPP Pin Configurations, and Associated Card Operating Mode .....	36
Table 7.3–4 - CIS Minimum Set of Tuples .....	37
Table 7.4–1 - M-Mode Power Supply DC Characteristics .....	42
Table 7.4–2 - Card Signal Types by Mode .....	43
Table 7.4–3 - DC Signal Requirements .....	43
Table 7.4–4 - DC Signaling Characteristics for the “LogicPC” Signaling Level .....	44
Table 7.4–5 - DC Signaling Characteristics for the “LogicCB” Signaling Level .....	44
Table 7.4–6 - CableCARD and Host Pullups and Pulldowns.....	45
Table 7.4–7 - S-Mode Signal Parameters.....	46
Table 7.4–8 - M-CARD Power-On and Reset Timing Requirements.....	49
Table 7.4–9 - M-CARD MPEG Transport Timing.....	49
Table 7.4–10 - M-Mode Serial Interface Timing.....	51
Table 7.6–1 - Extended Interface Registers .....	54
Table 7.6–2 - Control Register Definitions .....	55
Table 7.6–3 - Status Register Definitions .....	56
Table 7.6–4 - CPU Interface Packet Format.....	56
Table 7.6–5 - Length field used by all PDUs at Transport, Session and Application Layers .....	65
Table 7.6–6 - Expected Received Objects – Transport Connection on the Host.....	68
Table 7.6–7 - Expected Received Objects – Transport Connection on the Card .....	69
Table 7.6–8 - Command TPDU (C_TPDU).....	71
Table 7.6–9 - Response TPDU (R_TPDU) .....	72
Table 7.6–10 - Coding of bit8 of SB_value .....	72
Table 7.6–11 - Create Transport Connection (Create_T_C) .....	73
Table 7.6–12 - Create Transport Connection Reply (C_T_C_Reply) .....	73
Table 7.6–13 - Delete Transport Connection (Delete_T_C) .....	73
Table 7.6–14 - Delete Transport Connection Reply (D_T_C Reply).....	74
Table 7.6–15 - Request Transport Connection (Request_T_C ) .....	74
Table 7.6–16 - New Transport Connection (New_T_C).....	75

Table 7.6–17 - Transport Connection Error (T_C_Error) .....	75
Table 7.6–18 - Error Code Values .....	75
Table 7.6–19 - Send Data C_TPDU .....	76
Table 7.6–20 - Send Data R_TPDU .....	76
Table 7.6–21 - Receive Data C_TPDU .....	76
Table 7.6–22 - Receive Data R_TPDU .....	77
Table 7.6–23 - Transport Tag Values .....	77
Table 7.6–24 - Profile Changed .....	79
Table 7.6–25 - Profile Inquiry .....	79
Table 7.6–26 - Profile Reply .....	80
Table 7.6–27 - Buffer 1 .....	82
Table 7.6–28 - Buffer 2 .....	82
Table 7.6–29 - Buffer 3 .....	83
Table 7.6–30 - Buffer 4 .....	83
Table 9.1–1 - SPDU Structure Syntax .....	87
Table 9.1–2 - open_session_request() Syntax .....	87
Table 9.1–3 - open_session_response() Syntax .....	88
Table 9.1–4 - close_session_request() Syntax .....	88
Table 9.1–5 - close_session_response() Syntax .....	89
Table 9.1–6 - session_number() Syntax .....	89
Table 9.1–7 - Summary of SPDU Tags .....	89
Table 9.2–1 - Public Resource Identifier .....	90
Table 9.2–2 - Private Resource Identifier .....	90
Table 9.2–3 - resource_identifier() Syntax .....	91
Table 9.3–1 - APDU Structure Syntax .....	91
Table 9.3–2 - Resource Identifier Values .....	92
Table 9.3–3 - Application Object Tag Values .....	93
Table 9.3–4 - Host-Card Interface Resource Loading .....	96
Table 9.4–1 - Resource Manager Resource Identifier .....	97
Table 9.4–2 - Resource Manager APDU List .....	97
Table 9.4–3 - profile_inq() APDU Syntax .....	97
Table 9.4–4 - profile_reply() APDU Syntax .....	98
Table 9.4–5 - profile_changed() APDU Syntax .....	98
Table 9.5–1 - Application Information Resource Identifier .....	98
Table 9.5–2 - Application Information APDU List .....	99
Table 9.5–3 - application_info_req() APDU Syntax .....	99
Table 9.5–4 - application_info_cnf() APDU Syntax .....	101
Table 9.5–5 - server_query() APDU Syntax .....	102
Table 9.5–6 - server_reply() APDU Syntax .....	104
Table 9.6–1 - A Low Speed Communication Resource .....	104
Table 9.6–2 - Low-Speed Communication Resource ID Reporting Matrix .....	105
Table 9.7–1 - C A Support Resource .....	105
Table 9.7–2 - CA Support APDUs .....	106
Table 9.7–3 - ca_info_inquiry() APDU Syntax .....	106
Table 9.7–4 - ca_info() APDU Syntax .....	106
Table 9.7–5 - S-Mode ca_pmt() APDU Syntax (Resource Type 1 Version 2) .....	107
Table 9.7–6 - M-Mode ca_pmt() APDU Syntax (Resource Type 2 Version 1) .....	109
Table 9.7–7 - S-Mode ca_pmt_reply() APDU Syntax (Resource Type 1 Version 2) ...	113
Table 9.7–8 - M-Mode ca_pmt_reply() APDU Syntax (Resource Type 2 Version 1) ...	114
Table 9.7–9 - S-Mode ca_update() APDU Syntax (Resource Type 1 Version 2) .....	115
Table 9.7–10 - M-Mode ca_update() APDU Syntax (Resource Type 2 Version 1) .....	116



Table 9.8–1 - Host Control Support Resource .....	118
Table 9.8–2 - Host Control Support APDUs .....	118
Table 9.8–3 - OOB_TX_tune_req() APDU Syntax .....	118
Table 9.8–4 - RF TX Frequency Value .....	118
Table 9.8–5 - RF TX Power Level .....	119
Table 9.8–6 - RF TX Rate Value .....	119
Table 9.8–7 - OOB_TX_tune_cnf() APDU Syntax .....	119
Table 9.8–8 - OOB_RX_tune_req() APDU Syntax .....	120
Table 9.8–9 - RF RX Frequency Value .....	120
Table 9.8–10 - OOB Transmit Rate Format .....	120
Table 9.8–11 - OO_RX_tune_cnf() APDU Syntax .....	120
Table 9.8–12 - S-Mode - inband_tune_req() APDU Syntax (Resource Type 1 Version 3) .....	121
Table 9.8–13 - Tune Frequency Value .....	121
Table 9.8–14 - M-Mode - inband_tune_req() APDU Syntax (Resource Type 2 Version 1) .....	122
Table 9.8–15 - Tune Frequency Value .....	122
Table 9.8–16 - S-Mode - inband_tune_cnf() APDU Syntax (Resource Type 1 Version 3) .....	123
Table 9.8–17 - M-Mode - inband_tune_cnf() APDU Syntax (Resource Type 1 Version 3) .....	123
Table 9.10–1 - System Time Support Resource .....	124
Table 9.10–2 - System Time Support APDUs .....	124
Table 9.10–3 - Transmission of system_time_inq .....	124
Table 9.10–4 - system_time APDU (Type 1) .....	124
Table 9.10–5 - system_time APDU (Type 2) .....	125
Table 9.11–1 - MMI Support Resource .....	126
Table 9.11–2 - MMI Support APDUs .....	126
Table 9.11–3 - open_mmi_req() .....	126
Table 9.11–4 - S-Mode - open_mmi_cnf (Resource Type 2) .....	127
Table 9.11–5 - M-Mode - open_mmi_cnf (Resource Type 3) .....	127
Table 9.11–6 - close_mmi_req .....	128
Table 9.11–7 - close_mmi_cnf .....	128
Table 9.12–1 - CableCARD Device Resources Resource .....	129
Table 9.12–2 - CableCARD Resources Support APDUs .....	129
Table 9.12–3 - stream_profile APDU Syntax .....	130
Table 9.12–4 - stream_profile_cnf APDU .....	130
Table 9.12–5 - program_profile APDU .....	130
Table 9.12–6 - programs_profile_cnf APDU .....	131
Table 9.12–7 - es_profile APDU Syntax .....	131
Table 9.12–8 - es_profile_cnf APDU Syntax .....	131
Table 9.12–9 - request_pids APDU .....	132
Table 9.12–10 - request_pids_cnf APDU .....	132
Table 9.13–1 - CableCARD Copy Protection Resource .....	133
Table 9.14–1 - Extended Channel Support Resource .....	133
Table 9.14–2 - Extended Channel Support APDUs .....	134
Table 9.14–3 - new_flow_req APDU Syntax .....	136
Table 9.14–4 - Card DHCP Vendor Specific Information (Option 43) Sub-option Encoding .....	137
Table 9.14–5 - Card DHCP Vendor Class Identifier (Option 60) Encoding .....	138
Table 9.14–4 - new_flow_cnf APDU Syntax .....	139

Table 9.14–5 - Flag field definitions .....	140
Table 9.14–6 - delete_flow_req APDU Syntax .....	140
Table 9.14–7 - delete_flow_cnf APDU Syntax .....	140
Table 9.14–8 - lost_flow_ind APDU Syntax .....	141
Table 9.14–9 - lost_flow_cnf APDU Syntax .....	141
Table 9.14–10 - inquire_DSG_mode APDU Syntax .....	143
Table 9.14–11 - set_DSG_mode APDU Syntax .....	143
Table 9.14–12 - Configure Advanced DSG Object Syntax .....	147
Table 9.14–13 - DSG Message Object Syntax .....	149
Table 9.14–14 - send_DCD_info Object Syntax (Resource Version 3) .....	151
Table 9.14–15 - DSG_error APDU Syntax .....	151
Table 9.15–1 - Generic Feature Control Resource .....	154
Table 9.15–2 - Feature Ids .....	155
Table 9.15–3 - Generic Feature Control APDUs .....	155
Table 9.15–4 - feature_list_req APDU Syntax .....	156
Table 9.15–5 - feature_list APDU Syntax .....	156
Table 9.15–6 - feature_list_cnf APDU Syntax .....	156
Table 9.15–7 - feature_list_changed APDU Syntax .....	157
Table 9.15–8 - feature_parameters_req APDU Syntax .....	157
Table 9.15–9 - feature_parameters APDU Syntax .....	158
Table 9.15–10 - Feature Parameters Confirm Object Syntax .....	159
Table 9.15–11 - rf_output_channel .....	160
Table 9.15–12 - p_c_pin .....	160
Table 9.15–13 - p_c_settings .....	160
Table 9.15–14 - purchase_pin .....	161
Table 9.15–15 - time_zone .....	161
Table 9.15–16 - daylight_savings .....	162
Table 9.15–17 - ac_outlet .....	162
Table 9.15–18 - language .....	162
Table 9.15–19 - rating_region .....	163
Table 9.15–20 - reset_pin .....	163
Table 9.15–21 - cable_urls .....	163
Table 9.15–22 - EA_location_code .....	164
Table 9.16–1 - Generic Diagnostic Support Resource .....	164
Table 9.16–2 - Generic Diagnostic Support APDUs .....	165
Table 9.16–3 - Diagnostic Ids .....	165
Table 9.16–4 - S-Mode - diagnostic_req APDU Syntax (Version 1) .....	165
Table 9.16–5 - M-Mode - diagnostic_req APDU Syntax (Version 2) .....	166
Table 9.16–6 - S-Mode - diagnostic_cnf APDU Syntax (Version 1) .....	167
Table 9.16–7 - M-Mode - diagnostic_cnf APDU Syntax (Version 2) .....	169
Table 9.16–8 - Table Status Field Values .....	170
Table 9.16–9 - memory_report .....	170
Table 9.16–10 - software_ver_report .....	171
Table 9.16–11 - firmware_ver_report .....	171
Table 9.16–12 - MAC_address_report .....	172
Table 9.16–13 - FAT_status_report .....	172
Table 9.16–14 - FDC_status_report .....	173
Table 9.16–15 - FDC Center Frequency Value .....	173
Table 9.16–16 - current_channel_report .....	174
Table 9.16–17 - 1394_port_report .....	174
Table 9.16–18 - DVI Status Report Syntax .....	175

Table 9.16–19 - Frame Rate Associated With the Video Format On the DVI Link .....	176
Table 9.16–20 - Aspect Ratio Associated With the Video Format On the DVI Link.....	176
Table 9.16–21 - eCMStatus Report Syntax .....	177
Table 9.16–22 - Downstream Center Frequency Value.....	177
Table 9.16–23 - Upstream Transmit Center Frequency Value .....	177
Table 9.16–24 - HDMI Status Report Syntax.....	178
Table 9.16–25 - Frame Rate Associated With the Video Format On the HDMI Link....	178
Table 9.16–26 - Aspect Ratio Associated With the Video Format On the HDMI Link ..	179
Table 9.16–27 - RDC_status_report .....	179
Table 9.16–28 - RDC Center Frequency Value .....	179
Table 9.16–29 - net_address_report.....	180
Table 9.17–1 - Specific Application Support Resource.....	182
Table 9.17–2 - Specific Application Support APDUs .....	182
Table 9.17–3 - SAS_connect_rqst APDU Syntax .....	183
Table 9.17–4 - SAS_connect_cnf APDU Syntax .....	183
Table 9.17–5 - SAS_data_rqst APDU Syntax.....	184
Table 9.17–6 - SAS_data_av APDU Syntax.....	184
Table 9.17–7 - SAS_data_cnf APDU Syntax .....	184
Table 9.17–8 - SAS_server_query APDU Syntax.....	185
Table 9.17–9 - SAS_server_reply APDU Syntax.....	185
Table 9.17–10 - SAS_Async Message APDU Syntax .....	186
Table 9.18–1 - Homing Resource .....	191
Table 9.18–2 - Homing Objects .....	191
Table 9.18–3 - Open Homing Object Syntax .....	191
Table 9.18–4 - Open Homing Reply Object Syntax .....	192
Table 9.18–5 - Homing Active Object Syntax .....	192
Table 9.18–6 - Homing Cancelled Object Syntax .....	192
Table 9.18–7 - Homing Complete Object Syntax.....	192
Table 9.18–8 - Firmware Upgrade Object Syntax.....	193
Table 9.18–9 - Firmware Upgrade Reply Object Syntax.....	194
Table 9.18–10 - Firmware Upgrade Complete Object Syntax .....	194
Table 9.19–1 - Resource Identifier.....	201
Table 9.19–2 - Table of Application Protocol Data Units .....	201
Table 9.19–3 - host_info_request .....	202
Table 9.19–4 - host_info_response .....	202
Table 9.19–5 - Code Version Table .....	204
Table 9.19–6 - code_version_table_reply.....	205
Table 9.19–7 - host_download_control table .....	206
Table 9.19–8 - host_download_command.....	206
Table 10.4–1 - Flow Requirements.....	211
Table 10.6–1 - S-Mode Extended Channel Link Layer Packet .....	211
Table 10.6–2 - M-Mode Extended Channel Link Layer Packet .....	212
Table A–1 - HTML Keyword List .....	215
Table A–2 - Characters.....	216
Table B–1 - Error Handling .....	221
Table D.2–1 - CISTPL_LINKTARGET .....	235
Table D.2–2 - CISTPL_DEVICE_0A.....	236
Table D.2–3 - CISTPL_DEVICE_0C.....	236
Table D.2–4 - CISTPL_VERS_1 .....	237
Table D.2–5 - CISTPL_MANFID .....	238
Table D.2–6 - CISTPL_CONFIG.....	238

Table D.2-7 - CCST-CIF .....	239
Table D.2-8 - CISTPL_CFTABLE_ENTRY .....	239
Table D.2-9 - STCE_EV .....	240
Table D.2-10 - STCE_PD .....	241
Table D.2-11 - CISTPL_END .....	241
Table D.3-1 - Configuration Option Register .....	241
Table E-1 - DownloadInfoIndicator Message Detail .....	243

## Figures

Figure 5.2-1 - Card Interfaces .....	15
Figure 5.2-2 - Transport Stream Interface Layers .....	16
Figure 5.2-3 - Command Interface Layers .....	16
Figure 5.5-1 - System with One-way Network .....	18
Figure 5.6-1 - System with Two-way Network .....	19
Figure 5.7-1 - System with DOCSIS Two-way Network .....	20
Figure 5.8-1 - Host and M-CARD Device Block Diagram Example .....	21
Figure 5.10-1 - CableCARD Out-of-Band Interface .....	23
Figure 5.10-2 - M-Mode: CHI Diagram .....	24
Figure 5.10-3 - DSG Packet Format Across Card Interface .....	25
Figure 5.10-4 - S-Mode CHI Diagram .....	26
Figure 7.3-1 - Timing Relationships for Transport Stream Interface Signals .....	32
Figure 7.3-2 - Card Type Detection Signals .....	36
Figure 7.3-3 - CMP Diagram .....	39
Figure 7.3-4 - M-Mode MPEG Transport Stream Pre-Header .....	39
Figure 7.3-5 - CRC Polynomial .....	40
Figure 7.4-1 - CableCARD Device Output Timing Diagram .....	46
Figure 7.4-2 - CableCARD Device Input Timing Diagram .....	47
Figure 7.4-3 - M-CARD Power-On and Reset Timing Diagram .....	48
Figure 7.4-4 - M-Mode Serial Interface Timing Diagram .....	50
Figure 7.6-1 - Hardware Interface Registers .....	52
Figure 7.6-2 - Status Register .....	52
Figure 7.6-3 - Command Register .....	53
Figure 7.6-4 - Modem in-the-Card System Overview .....	54
Figure 7.6-5 - Modem in-the-Host System View .....	54
Figure 7.6-6 - Map of Hardware Interface Registers .....	55
Figure 7.6-7 - Card RS Operation .....	60
Figure 7.6-8 - CableCARD Personality Change Sequence .....	63
Figure 7.6-9 - Layout of Link Protocol Data Unit .....	65
Figure 7.6-10 - State Transition Diagram – Host Side of the Transport Protocol .....	68
Figure 7.6-11 - State Transition Diagram – Card Side of the Transport Protocol .....	69
Figure 7.6-12 - Object Transfer Sequence – Transport Protocol .....	70
Figure 7.6-13 - C_TPDU Structure .....	71
Figure 7.6-14 - R_TPDU Structure .....	71
Figure 7.6-15 - SB_value .....	72
Figure 7.6-16 - Create_T_C Structure .....	73
Figure 7.6-17 - C_T_C_Reply Structure .....	73
Figure 7.6-18 - Delete_T_C Structure .....	74
Figure 7.6-19 - D_T_C_Reply Structure .....	74
Figure 7.6-20 - Request_T_C Structure .....	74
Figure 7.6-21 - Request_T_C Structure .....	75

Figure 7.6-22 - T_C_Error Structure .....	75
Figure 7.6-23 - Send Data command/ Response Pair .....	76
Figure 7.6-24 - Receive Data command/ Response Pair .....	76
Figure 7.6-25 - Object Transfer Sequence – Transport Protocol .....	78
Figure 7.6-26 - CableCARD Device Interrupt Logical Operation .....	81
Figure 7.6-27 - M-Mode Serial Interface Protocol Diagram .....	84
Figure 9.1-1 - SPDU Structure .....	86
Figure 9.3-1 - APDU Structure .....	91
Figure 9.3-2 - Primitive Tag Coding .....	92
Figure 9.7-1 - Program Index Table 1 .....	111
Figure 9.7-2 - Program Index Table 2 .....	111
Figure 9.7-3 - Program Index Table 3 .....	112
Figure 9.14-1 - DSG Mode Message Flow .....	142
Figure 9.14-2 - Sample Advanced Mode Message Flow .....	146
Figure 9.15-1 - Generic Feature List Exchange .....	152
Figure 9.15-2 - Card Feature List Change .....	153
Figure 9.15-3 - Host Feature List Change .....	153
Figure 9.15-4 - Host to CableCARD Device Feature Parameters .....	153
Figure 9.15-5 - Host Parameter Update .....	154
Figure 9.15-6 - Headend to Host Feature Parameters .....	154
Figure 9.17-1 - Specific Application Support Connection Sequence .....	181
Figure 9.17-2 - Specific Application Support Alternate Connection Sequence .....	182
Figure 9.18-1 - Firmware Upgrade Flowchart .....	190
Figure 9.19-1 - One-Way Operation .....	197
Figure 9.19-2 - Two-Way Operation– Command .....	198
Figure 9.19-3 - Two-Way Operation – On-Demand .....	199
Figure 9.19-4 - Flow Chart Summarizing Download Operations .....	200
Figure 10.2-1 - Flow Examples - QPSK Modem Case .....	209
Figure 10.3-1 - Flow Examples - eCM Case .....	210
Figure B–1 - Error Display .....	233
Figure C–1 - 8 bit CRC generator/checker model .....	234

This page intentionally left blank.

# 1 SCOPE

This specification defines the normative characteristics for the interface between a security module owned and distributed by cable operators and commercially available consumer receivers and set-top terminals, “Host Devices”, that are used to access multi-channel television programming delivered on North American cable systems. Some examples of the Host devices could be a set-top box, a television, a VCR, etc. In some cases the local cable operator may optionally choose to supply this Host device in addition to the security module. In order to receive scrambled cable services, the Host would require this security module, called a CableCARD™ device, to be inserted and authorized to receive services. This CableCARD device, previously identified as a Point of Deployment (POD) module, provides the conditional access operation and the network connectivity for the Host.

This CableCARD device-Host Interface (CHI) specification defines the interface between the Host device (Host) and the CableCARD device (Card).

There are currently two modes of operation in which the Host and the Card can operate. The Single-Stream CableCARD device (S-CARD) is the first generation security module that can only operate in the Single-Stream Mode (S-Mode), and the Multi-Stream CableCARD device (M-CARD), a second-generation variant, is capable of operation in Multi-Stream Mode (M-Mode), or in Single-Stream Mode (S-Mode), based on the Host and its available functionality.

This document defines the interface for both the S-CARD and the M-CARD and the different operating modes. The M-CARD, when operating in S-Mode, is backward compatible with the Single-Stream CableCARD Host Interface as previously defined via ANSI/SCTE 28, the Host-POD Interface Standard, and ANSI/SCTE 41, the POD Copy Protection System. When the M-CARD is running in M-Mode, functionality to support multiple program decryption from multiple transport streams is added. One application for the M-CARD could be a Host device with multiple tuners and QAM demodulators.

While analog television channels may be tuned, only digital television channels will be passed through the Card for descrambling of authorized conditional access channels, and passed back to the Host. The Card will not only provide the conditional access decryption of the digital television channel, but MAY also provide the network interface between the Host and the cable system.

This document is a compilation of the specifications, standards, and related text from the OpenCable Specification CableCARD Interface documents, single stream and multi-stream, OC-SP-CC-IF and OC-SP-MC-IF, as well as the SCTE 28, 2004 documents. For the text that was extracted from the SCTE 28 2004 document, in all cases, the terms “POD” and “POD module” were replaced with the terms “Card” and “CableCARD device”.

## 1.1 Introduction and Overview

This specification defines the characteristics and normative specifications for the interface between the Card device and the Host device. This specification describes the interface for both the Single Stream Card and the Multi-Stream Card.

1. Single Stream Card (S-CARD) for use with or between Cards and Hosts capable only of processing a single program on the interface is said to be operating in Single Stream Mode (S-Mode).
2. Multi-Stream Card (M-CARD) for use between a Card and Host both capable of processing multiple simultaneous programs on the interface is said to be operating in Multi-Stream Mode (M-Mode). The M-CARD, when instructed to do so by the Host, can operate in S-Mode. The M-CARD can only operate in either M-Mode or S-Mode, not both.

This specification supports a variety of conditional access scrambling systems. Entitlement management messages (EMMs) and Entitlement Control Messages (ECMs) across the interface for such scrambling systems are carried in the cable out-of-band channel as defined by [SCTE55-2], [SCTE55-1], and the DOCSIS® Set-top Gateway Specification [DSG].

The interface will support Emergency Alert messages transmitted over the out-of-band channel to the Card which will deliver the message to the Host using the format defined in [J042].

This specification defines, sometimes by reference, the physical interface, signal timing, link interface, and application interface for the Card-Host interface (CHI).

## 1.2 Historical Perspective (Informative)

Portions of this specification have origins in EIA-679, the National Renewable Security Standard, which was initially adopted in September 1998. Part B of that standard uses the same physical size, shape and connector of the computer industry PCMCIA card defined elsewhere, and defines the interface protocols and stack. Part B of that standard was adopted by SCTE DVS.

Further extensions and modifications of EIA-679 led to the adoption of EIA-679-B in 2000. The EIA-679 standard was developed substantially by the EIA/NCTA Joint Engineering Committee (JEC) National Renewable Security Standard (NRSS) Subcommittee, and was a joint work of NCTA and CEMA Technology & Standards.

The M-Mode specification has its origin in ANSI/SCTE 28, where the original version of the Card provided only enough bandwidth for a S-Mode. As DVRs, picture-in-picture, and other M-Mode features were developed, it was realized that the original S-Mode had inadequate bandwidth for some of these features, and could not grow to support multi-tuner gateway scenarios.

A M-Mode provides the higher transport data throughput rates that would be required to support future features, such as multiple-tuner Hosts, Hosts with DVRs, Hosts with picture-in-picture capability, and future extensions of existing conditional access functions to include Digital Rights Management.

This specification document is based on and conforms to much of the technical content as found in ANSI/SCTE 28, 2004.

## 1.3 Requirements (Conformance Notation)

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

“SHALL/MUST”	These words or the adjective “REQUIRED” means that the item is an absolute requirement of this specification.
“SHALL NOT/ MUST NOT”	This phrase means that the item is an absolute prohibition of this specification.
“SHOULD”	This word or the adjective “SHOULD” means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
“SHOULD NOT”	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
“MAY”	This word or the adjective “MAY” means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.



## 1.4 Numerical

In most cases all numbers without a prefix are base 10 (decimal). The following prefixes are to be used to designate different bases.

**Table 1.4–1 - Numerical Representation**

Prefix/Suffix	Base	Name
b	2	Binary
0	10	Decimal
0x	16	Hexadecimal

## 2 REFERENCES

The following specifications and standards contain provisions that, through reference in this text, constitute normative provisions of this specification. At the time of publication, the editions indicated are current. All standards are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying for the most recent editions of the standards listed in this section.

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

### 2.1 Normative References

[DSG]	DOCSIS® Set-top Gateway (DSG) Interface Specification, SP-DSG-I04-050408, April 8, 2005, Cable Television Laboratories, Inc.
[DOCSIS2.0]	Data-Over-Cable Service Interface Specification DOCSIS 2.0, OC-SP-RFIV2.0-I08-050408, April 8, 2005, Cable Television Laboratories, Inc.
[OCHD2]	OpenCable Host Device 2.0 Core Functional Requirements, OC-SP-HOST2.0-CFR-I06-050708.
[J042]	American National Standard, J-STD-042-2002, Emergency Alert Message for Cable (SCTE 18 and EIA/CEA 814).
[ISO10646-1]	ISO/IEC 10646-1: 1993 Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane.
[ISO13818-1]	ISO/IEC 13818-1 Generic Coding of Moving Pictures and Associated Audio: Systems.
[ISO13818-6]	ISO/IEC 13818-6 Op Cit, Extensions for DSM-CC.
[ISO13818-9]	ISO/IEC 13818-9 Extension for real time interface for systems decoders.
[ISO639-1]	ISO 639-1: 2002 Codes for the representation of names of Languages - Part 1: Alpha-2 code.
[ISO639-2]	ISO 639-2: 2002 Codes for the representation of names of Languages - Part 1: Alpha-3 code.
[ISO8825]	ISO 8825: 1987 Open Systems Interconnection- Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)
[ISO8859-1]	ISO 8859-1: 1998 Information technology, 8-bit single-byte coded graphic character sets, Part 1: Latin alphabet No. 1.
[CCCP2]	OpenCable™ CableCARD™ Copy Protection 2.0 Specification, OC-SP-CCCP2.0-IF-I02-050708, July 8, 2005, Cable Television Laboratories, Inc.
[OCAP]	OpenCable™ Application Platform Specification, OCAP 1.0 Profile, OC-SP-OCAP1.0-I15-050415, April 15, 2005, Cable Television Laboratories, Inc.
[OCCD]	OpenCable Common Download Specification, OC-SP-CD-IF-I08-040831.
[PCMCIA2]	PCMCIA PC Card Standard Volume 2 Release 8.0, April 2001 Electrical Specification.
[PCMCIA3]	PCMCIA PC Card Standard Volume 3 Release 8.0, April 2001 Physical Specification.
[PCMCIA4]	PCMCIA PC Card Standard Volume 4 Release 8.0, April 2001 Metaformat Specification.
[RFC2132]	DHCP Options and BOOTP Vendor Extensions, March 1997.

[RFC2396]	IETF RFC 2396, August 1998, T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax".
[RFC2131]	IETF RFC 2131, March 1997, R. Droms, "Dynamic Host Configuration Protocol".
[SCTE23]	ANSI/SCTE 23-2 2002 is DOCSIS 1.1 Part 2 Baseline Privacy Interface Plus.
[SCTE28]	ANSI/SCTE 28 2004 Host POD Interface Standard.
[SCTE41]	ANSI/SCTE 41 2004 POD Copy Protection System.
[SCTE55-1]	ANSI/SCTE 55-1 2002 Digital Broadband Delivery System: Out Of Band Transport Part 1: Mode A.
[SCTE55-2]	ANSI/SCTE 55-2 2002 Digital Broadband Delivery System: Out Of Band Transport Part 2: Mode B.
[SCTE65]	ANSI/SCTE 65 2002 Service Information Delivered Out Of Band.
[SCTE80]	ANSI/SCTE 80 2002 In-Band Data Broadcast Standard including Out-of-Band Announcements.

## 2.2 Informative References

[CHILA]	CableLabs CableCARD-Host Interface License Agreement.
[NRSSB]	EIA-679-B Part B, National Renewable Security Standard. A joint work of NCTA and CEMA Technology and Standards.

## 2.3 Reference Acquisition

- ***CableLabs Specifications and License Agreements***

Cable Television Laboratories, Inc. 858 Coal Creek Circle , Louisville, CO 80027;  
Telephone: +1-303-661-9100; Internet: <http://www.cablelabs.com/>

- ***ISO/IEC Specifications***

ISO Central Secretariat: International Organization for Standardization (ISO), 1, rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland; Internet: <http://www.iso.ch/>

- ***SCTE/DVS Specifications***

SCTE - Society of Cable Telecommunications Engineers Inc., 140 Philips Road, Exton, PA 19341;  
Telephone: 610-363-6888 / 800-542-5040; Fax: 610-363-5898; Internet: <http://www.scte.org/>

- ***ANSI/EIA Standards***

American National Standards Institute, Customer Service, 11 West 42<sup>nd</sup> Street, New York, NY 10036;  
Telephone 212-642-4900; Facsimile 212-302-1286; E-mail [sales@ansi.org](mailto:sales@ansi.org); URL <http://www.ansi.org>

- ***EIA Standards: United States of America***

Global Engineering Documents, World Headquarters, 15 Inverness Way East, Englewood, CO USA 80112-5776; Telephone: 800-854-7179; Facsimile: 303-397-2740; E-mail: [global@ihs.com](mailto:global@ihs.com); URL: <http://global.ihs.com>

- ***ITU Standards***

ITU Sales and Marketing Service, International Telecommunication Union, Place des Nations CH-1211, Geneva 20, Switzerland; Telephone: +41 22 730 6141; Facsimile: +41 22 730 5194; E-mail: [sales@itu.int](mailto:sales@itu.int) ; URL: <http://www.itu.org>

- ***Internet Specifications***

The Internet Engineering Task Force, IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20101-5434; Telephone: 703-620-8990; Facsimile: 703-620-9071; E-mail: [ietf-secretariat@ietf.org](mailto:ietf-secretariat@ietf.org); URL: <http://www.ietf.org/rfc>

### 3 TERMS AND DEFINITIONS

This specification uses the following terms:

Term	Definition
<b>American Standard Code for Information Interchange</b>	Internationally recognized method for the binary representation of text.
<b>Application Protocol Data Unit</b>	A common structure to send application data between the Card and Host.
<b>Application Program Interface</b>	The software interface to system services or software libraries. An API can consist of classes, function calls, subroutine calls, descriptive tags, etc.
<b>CableCARD™ device</b>	A PCMCIA card distributed by cable providers and inserted into a Host device to enable premium services in compliance with the OpenCable specifications, also called “Card” and “Point of Deployment” (POD) module.
<b>Card</b>	CableCARD device
<b>Card Information Structure</b>	Low-level configuration information contained in the Card’s Attribute Memory.
<b>Command Channel</b>	Also identified as Data Channel.
<b>Conditional Access and encryption</b>	A system that provides selective access to programming to individual customers.
<b>Conditional Access System</b>	Secures delivery of cable services to the Card.
<b>CPU Interface</b>	The logical interface between the Card and the Host comprised of the Data and Extended communications channels.
<b>Data Channel</b>	Also identified as Command Channel.
<b>DOCSIS Set-top Gateway</b>	A method of using DOCSIS protocols to support a one-way out-of-band communication path.
<b>Downstream</b>	Transmission from headend to Host.
<b>DSG Advanced Mode</b>	Operation with the DCD message. Address assignment is dynamic. The DSG Tunnel Address is determined by the DSG Agent and learned by the DSG Client through the DSG Address Table in the DCD message.
<b>DSG Basic Mode</b>	Operation without the DCD message. Address assignment is static. The DSG Tunnel Address is determined by the DSG Client and learned by the DSG Agent through configuration. This mode provides backwards compatibility with earlier versions of the DSG specification.
<b>DSG Tunnel</b>	A stream of packets sent from the CMTS to the eCM. In DSG Basic Mode, a DSG Tunnel is identified solely by its DSG Tunnel Address; all of a DSG Tunnel’s packets use the same DSG Tunnel Address and different DSG Tunnels use different DSG Tunnel Addresses. In DSG Advanced Mode, a DSG Tunnel might be identified solely by its DSG Tunnel Address, or it might be identified by a combination of the DSG Tunnel Address along with other DSG Rule parameters: UCID range, Classifier IP addresses, and TCP port numbers.

Term	Definition
<b>Dynamic Host Configuration Protocol</b>	An Internet standard for assigning IP addresses dynamically to IP hosts.
<b>eCM</b>	A DOCSIS Cable Modem that has been embedded into a Set-top/Host device and includes DSG functionality.
<b>Encryption Mode Indicator</b>	Defines the copy protection mode for digital outputs.
<b>Entitlement Management Message</b>	A conditional access control message to a Card.
<b>Extended Application Information Table</b>	Used for launching and managing the lifecycle of unbound applications for OCAP.
<b>Extended Text Table</b>	An MPEG 2 table contained in the Program and System Information Protocol ("PSIP"), which provides detailed descriptions of virtual channels and events.
<b>Forward Application Transport</b>	A data channel carried from the headend to the set-top or Host device in a modulated channel at a rate of 27 or 36 Mbps. MPEG-2 transport is used to multiplex video, audio, and data into the FAT channel.
<b>Forward Data Channel</b>	An out-of-band ("OOB") data channel from the headend to the Host.
<b>Gapped Clock</b>	A periodic signal in which some transitions are omitted creating gaps in a clock pattern.
<b>Headend</b>	The control center of a cable television system, where incoming signals are amplified, converted, processed and combined into a common cable along with any original cable casting, for transmission to subscribers. The System usually includes antennas, preamplifiers, frequency converters, demodulators, modulators, processors and other related equipment.
<b>High-Z</b>	Greater than 100K Ohm resistance to power or to ground.
<b>Host</b>	The consumer device used to access and navigate cable content. Typically a digital TV or set-top DTV receiver, further defined in [OCHD2].
<b>HTTP</b>	The transport layer for HTML documents over the Internet Protocol ("IP").
<b>Hypertext Markup Language</b>	A presentation language for the display of multiple media contents, typically used on the Internet.
<b>Inband</b>	Within the main Forward Applications Transport channel.
<b>Internet Protocol</b>	The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses.
<b>IP Unicast</b>	Point-to-Point Internet Protocol datagram service.
<b>IP Multicast</b>	Point to multi-point Internet Protocol datagram service.
<b>Local Transport Stream ID</b>	Assigned by the Host operating in M-Mode.
<b>LSB</b>	Least Significant Bit or Byte of a specified binary value.
<b>M-CARD</b>	Multi-Stream Card, capable of operating in either S-Mode or M-Mode.

Term	Definition
<b>M-Host</b>	Multi-Stream/Multi-Tuner Host device.
<b>M-Mode</b>	A Multi-Stream Card (M-CARD), capable of processing multiple simultaneous programs on the interface, is said to be operating in Multi-Stream Mode (M-Mode).
<b>Model ID</b>	The model as it is marketed and appears on the label of the Certified Device, and as reported in the Digital Certificate Usage Report (see OpenCable Host 2.0 Digital Certificate authorization Agreement).
<b>MPEG</b>	Moving Picture Experts Group. Colloquial name for ISO-IEC SC29/WG11, which develops standards for compressed full-motion video, still image, audio, and other associated information.
<b>MPEG-2 Video</b>	ISO-IEC 13818-2, international standard for the compression of video.
<b>MPEG-2 Transport</b>	ISO-IEC 13818-1, international standard for the transport of compressed digital media.
<b>MSB</b>	Most Significant Byte or Bit, of a specified binary value.
<b>National Television Systems Committee</b>	An entity that developed the analog television system used in North America and elsewhere.
<b>Out-of-Band</b>	The combination of the Forward and Reverse Data Channels. The OOB channel provides a data communication channel between the cable system and the Host.
<b>PC Card</b>	A device that complies with the PC Card Standard, as referenced in this document.
<b>Point of Deployment</b>	Synonymous with “POD”, “point of deployment module”, CableCARD device and Card. A detachable device distributed by cable providers and inserted into a Host connector to enable reception of encrypted services.
<b>Protocol Data Unit</b>	A packet of data passed across a network or interface.
<b>Quadrature Amplitude Modulation</b>	A digital modulation method in which the value of a symbol consisting of multiple bits is represented by amplitude and phase states of a carrier. Typical types of QAM include 16-QAM (four bits per symbol), 32-QAM (five bits), 64QAM (six bits), and 256QAM (eight bits).
<b>Quadrature Phase Shift Keying</b>	A digital modulation method in which the state of a two-bit symbol is represented by one of four possible phase states.
<b>Remote Procedure Call</b>	The ability for client software to invoke a function or procedure call on a remote server machine.
<b>Resource</b>	A unit of functionality provided by the host for use by a Card. A resource defines a set of objects exchanged between Card and host by which the Card uses the resource.
<b>Return Data Channel</b>	A data communication channel running upstream from home to the headend, i.e., an out-of-band (“OOB”) data channel from the host to the headend.
<b>S-CARD</b>	Single-Stream Card compliant to ANSI/SCTE 28, capable of only operating in S-Mode.
<b>S-Host</b>	Single-Stream Host device.

Term	Definition
<b>S-Mode</b>	A Single Stream Card (S-CARD), capable only of processing a single program on the interface, is said to be operating in Single Stream Mode (S-Mode).
<b>Subtuple</b>	Subset of a Tuple.
<b>Tuple</b>	Data stored within a PC Card that can be used to determine the capabilities of the card.
<b>Uniform Resource Locator</b>	A standard method of specifying the location of an object or file.
<b>Upstream</b>	Transmission from host to headend.
<b>User Datagram Protocol</b>	A protocol on top of IP that is used for end-to-end transmission of user messages. Unlike TCP, UDP is an unreliable protocol, which means that it does not contain any retransmission mechanisms.
<b>Virtual Channel Table</b>	An MPEG-2 table which contains a list of all the channels that are or will be on plus their attributes.



## 4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations and acronyms:

Abbreviation/Acronym	Definition
<b>AES</b>	Advanced Encryption Standard
<b>ANSI</b>	American National Standards Institute
<b>APDU</b>	Application Protocol Data Unit
<b>API</b>	Application Program Interface
<b>ASCII</b>	American Standard Code for Information Interchange
<b>ASD</b>	Authorized Service Domain
<b>ATSC</b>	Advanced Television System Committee
<b>bslbf</b>	Bit String (serial) – Left Most Bit First
<b>CA</b>	Conditional Access
<b>CAS</b>	Conditional Access System
<b>CEA</b>	Consumer Electronic Association
<b>CHI</b>	Card - Host Interface
<b>CIS</b>	Card Information Structure
<b>CMOS</b>	Complementary Metal Oxide Silicon
<b>CMP</b>	CableCARD MPEG Packet
<b>CMTS</b>	Cable Modem Termination System
<b>CPU</b>	Central Processing Unit.
<b>CRC</b>	Cyclic Redundancy Check
<b>CVDT</b>	Code Version Download Table
<b>CVT</b>	Code Version Table
<b>DCD</b>	Downstream Channel Descriptor
<b>DHCP</b>	Dynamic Host Configuration Protocol
<b>DII</b>	Download Info Indicator
<b>DOCSIS®</b>	Data Over Cable Service Interface Specifications
<b>DRAM</b>	Dynamic Random Access Memory
<b>DSG</b>	DOCSIS Set-top Gateway
<b>DSM-CC</b>	Digital Storage Medium – Command and Control
<b>DVR</b>	Digital Video Recorder
<b>DVS</b>	Digital Video Subcommittee
<b>EAS</b>	Emergency Alert System
<b>ECM</b>	Entitlement Control Message
<b>eCM</b>	Embedded Cable Modem
<b>EIA</b>	Electronics Industries Alliance
<b>EMI</b>	Encryption Mode Indicator
<b>EMM</b>	Entitlement Management Message

<b>Abbreviation/Acronym</b>	<b>Definition</b>
<b>ETT</b>	Extended Text Table
<b>FAT</b>	Forward Application Transport
<b>FCC</b>	Federal Communications Commission
<b>FDC</b>	Forward Data Channel
<b>HTML</b>	Hypertext Markup Language
<b>I/O</b>	Input or output
<b>IB</b>	Inband
<b>ID</b>	Identifier/Identity/Identification
<b>IIR</b>	Initialize Interface Request
<b>IP</b>	Internet Protocol
<b>IP_U</b>	IP Unicast
<b>IP_M</b>	IP Multicast
<b>IPG</b>	Interactive Program Guide
<b>IPPV</b>	Impulse Pay-Per-View
<b>IQB</b>	Interface Query Byte
<b>kHz</b>	kilohertz
<b>LPDU</b>	Link Protocol Data Unit
<b>LTSID</b>	Local Transport Stream ID
<b>mA</b>	milliAmps
<b>MAC</b>	Media Access Control
<b>MHz</b>	Megahertz
<b>MMI</b>	Man-Machine Interface
<b>MoCA</b>	Multimedia over Coax Alliance
<b>ms</b>	millisecond
<b>MSO</b>	Multiple System Operator
<b>MTU</b>	Maximum Transmission Unit
<b>NAT</b>	Network Address Translation
<b>ns</b>	nanosecond
<b>NTSC</b>	National Television Systems Committee
<b>NVM</b>	Non-Volatile Memory
<b>OCAP</b>	OpenCable Application Platform
<b>OOB</b>	Out-of-Band
<b>OUI</b>	Organizationally Unique Identifier
<b>OCHD2</b>	OpenCable Host Device 2 (includes OCS2 and OCS2 Profiles)
<b>OCS2</b>	OpenCable Set-top 2
<b>PCMCIA</b>	Personal Computer Memory Card International Association
<b>PCR</b>	Program Clock Reference
<b>PDU</b>	Protocol Data Unit

<b>Abbreviation/Acronym</b>	<b>Definition</b>
<b>pF</b>	PicoFarad
<b>PHY</b>	Physical Layer
<b>PID</b>	Packet Identifier
<b>PIN</b>	Personal Identification Number
<b>PNG</b>	Portable Network Graphics
<b>POD</b>	Point of Deployment
<b>PPV</b>	Pay-Per-View
<b>PSI</b>	Program Specific Information
<b>QAM</b>	Quadrature Amplitude Modulation
<b>QPSK</b>	Quadrature Phase Shift Keying
<b>RDC</b>	Return Data Channel
<b>RF</b>	Radio Frequency
<b>RFC</b>	Request For Comments
<b>RPC</b>	Remote Procedure Call
<b>ROM</b>	Read Only Memory
<b>RX</b>	Receive
<b>SAS</b>	Specific Application Support
<b>SCTE</b>	Society of Cable Telecommunications Engineers
<b>SI</b>	System Information
<b>SRAM</b>	Static Random Access Memory
<b>tcimbsf</b>	Two's complement integer, (msb) sign bit first
<b>TPDU</b>	Transport Protocol Data Unit
<b>TSID</b>	Transport Stream Identifier
<b>TX</b>	Transmit
<b>UDP</b>	User Datagram Protocol
<b>uimsbf</b>	Unsigned Integer Most Significant Bit First
<b>URL</b>	Uniform Resource Locator
<b>V</b>	Volt
<b>VCR</b>	Video Cassette Recorder
<b>VCT</b>	Virtual Channel Table
<b>VOD</b>	Video-on-Demand
<b>XAIT</b>	Extended Application Information Table

## 5 MODEL OF OPERATION

The Card provides the conditional access operation and the network connectivity for the Host. MPEG transport streams are received by the Host and passed to the Card for decryption. The streams are returned to the Host device to be displayed.

In addition to MPEG streams, the CHI also carries out-of-band communications, as well as command and control signals. Cable system deployments utilize the OOB FDC and RDC paths for reception of the EMMs, SI data, EAS, and network connectivity.

### 5.1 Advanced Cable Services

The Card interface specification is designed to support advanced digital cable services by a digital television receiver when a Card is inserted.

In this case, “Advanced Digital Cable Services” would include support of the following functions:

- Interactive Program Guide
- Impulse Pay-Per-View (IPPV) using OCAP
- Video On Demand (VOD)
- Interactive Services

#### 5.1.1 Interactive Program Guide (IPG)

The Host may support an Interactive Program Guide (IPG) to enable the user to navigate to available services. The services supported by the IPG may include basic channel, premium channels, and Impulse Pay-Per-View (IPPV) events. Program guide data may be delivered to the application by means of the in-band (QAM) channel, DSG or FDC:

- In-band transmission of program and system information typically describes only the digital multiplex in which it is sent. This means that a single-tuner Host will periodically scan through all channels to receive data for each channel and store this information in memory.
- Optionally, at the discretion of the cable operator, the FDC or DSG may be used to deliver guide data. The format of this information over the FDC or DSG will be defined by the cable operator and may be used to support specific IPG implementations. The Host receives data from the Card over the **Extended Channel** as described in Section 10 of this document. This guide data typically describe the entire range of services offered by the cable system.

#### 5.1.2 Impulse Pay-Per-View (IPPV)

The Host may support the purchase of Impulse Pay-Per-View (IPPV) events using OCAP. The CableCARD Interface support of the Generic IPPV resource is deprecated. If supported, it SHALL comply with Section 8.10 of SCTE 28.

#### 5.1.3 Video-on-Demand (VOD)

Video-on-Demand (VOD) may be modeled as an IPPV event where the program stream is dedicated to an individual subscriber. The VOD application executes in the Host and supports all of the User Interface (UI) functions.

The additional streaming media control functions (i.e., Pause, Play, Fast-Forward, Rewind) may be supported using DSM-CC User-to-User messages. The **Extended Channel**, described in Section 10 of this document, may be used as the communication path for VOD signaling, and may also be used for VOD event purchases. After a VOD control session is established via the session creation interface, UDP messages may be exchanged transparently between the Host and the cable system. RFC 1831, 1832, and 1833 may be used as the underlying RPC mechanism for the exchange of DSM-CC UU.

### 5.1.4 Interactive services

Interactive Services may be supported by applications executing on the Host, such as, an email or game application. To advertise interactive services, a mechanism is required to deliver information about applications to the Host and the protocols described in [SCTE80] may be used for this purpose. Typically, information about interactive services are not associated with a streaming media service, so information about them is delivered via the FDC or DSG. The service information is passed to the Host via the **Extended Channel** resource when the Card serves as the OOB modem or across a DSG tunnel.

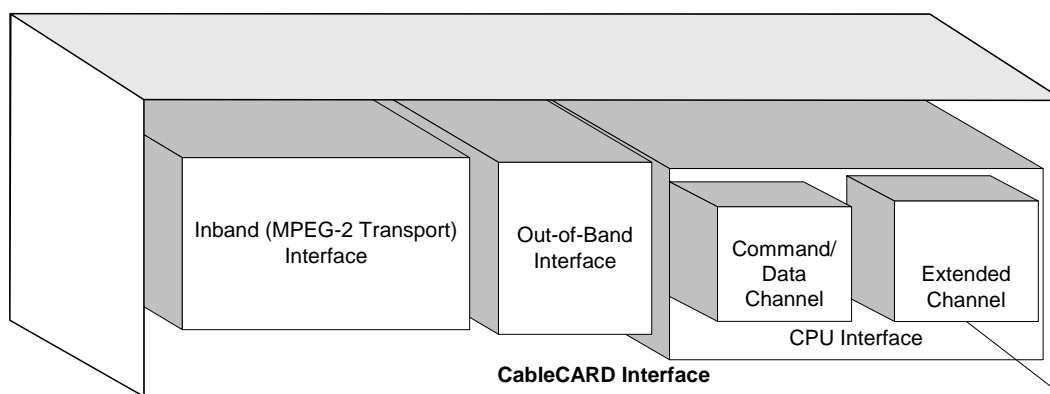
The **Extended Channel** may also be used as the communication path for interactive service signaling when the Card is serving as the OOB modem. After an interactive service session is established via the session creation interface, UDP messages may be exchanged transparently between the Host and the cable system. RFC 1831, 1832, and 1833 may be used as the underlying RPC mechanism for the exchange of application level messages.

## 5.2 CableCARD Device Functional Description

The CHI contains three sub-interfaces:

- An Inband interface for MPEG-2 Transport Stream input and output
- An Out-of-band Interface for receiving OOB data under two different delivery methods (but not simultaneously)
  - ANSI/SCTE 55-1 2002 (formerly DVS 178) Digital Broadband Delivery System: Out Of Band Transport Part 1: Mode A
  - ANSI/SCTE 55-2 2002 (formerly DVS 167) Digital Broadband Delivery System: Out Of Band Transport Part 2: Mode B
- A CPU Interface supporting:
  - Command Channel (also called Data Channel)
  - Extended Channel

The various interfaces are summarized in the following figure:



**Figure 5.2-1 - Card Interfaces**

Copy protection is required for protection of high-valued content, content marker with a non-zero EMI, across the CHI. Section 8, [CCCP2], as well as [SCTE41], identify this functionality and the expected behavior.

### 5.2.1 Transport Stream Interface

The in-band transport stream interface carries MPEG-2 transport packets in both directions. If the Card gives access to any services in the transport stream and those services have been selected by the Host, then the packets carrying those services will be returned descrambled, and the other packets are not modified. On the transport stream interface a constant delay through the module and any associated physical layer conditioning logic is preserved

under most conditions (see Section 7.3.6.1). The transport stream interface layers are shown below. The Transport Layer and all upper layers are defined in the MPEG-2 specification, ISO 13818.

Upper Layers
Transport Layers
PC Card Link Layer
PC Card Physical Layer

**Figure 5.2-2 - Transport Stream Interface Layers**

### 5.2.2 Command Interface

The Command Interface carries all the communication between the application(s) running in the Card and the Host. The communication protocols on this interface are defined in several layers in order to provide the necessary functionality. This functionality includes the ability to support complex combinations of transactions between the Card and host, and an extensible set of functional primitives (objects) which allow the host to provide resources to the Card. This layering is shown below.

Application			
Resources:			
User Interface	Low-Speed Communications	System	Optional extensions
Session Layer			
Generic Transport Sublayer			
PC Card Transport Sublayer			
PC Card Link Layer			
PC Card Physical Layer			

**Figure 5.2-3 - Command Interface Layers**

The PC Card implementation described has its own physical and link layers, and also its own transport lower sublayer. A future different physical implementation may differ in these layers and any difference will be restricted to these layers. The implementation-specific features of the transport lower sublayer are limited to coding and specific details of the message exchange protocol, and the common upper sublayer defines identification, initiation and termination of transport layer connections. The Session, Resource and Application layers are common to all physical implementations.

## 5.3 Network Connectivity/OOB Signaling

One of three types of signaling MAY be utilized, legacy OOB ([SCTE55-2] or [SCTE55-1]) or [DSG].

In the legacy OOB modes, the signaling functions are split between the Host and the Card such that only the RF processing and QPSK demodulation and modulation are done in the Host. The remainder of the processing, including all of the Data-link and MAC protocols, is implemented in the Card.

Hosts that only support the legacy OOB signaling methods can be either one-way or two-way Hosts. One-way Hosts lack the upstream transmitter of the legacy signaling method. DSG Hosts are all two-way Hosts.

In the DSG mode of operation, all of the Data-link and MAC level protocols are implemented in the embedded cable modem in the Host. In this case, the Card is not responsible for implementing these protocols, since they are provided via the embedded cable modem (eCM). The forward data channel messaging is transported as follows:

The forward data channel messaging is transported via one or more DSG Tunnels to the Host. The Host filters the IP packets on the DSG Tunnels identified by the Ethernet MAC addresses, specified by the Card. The Host, when instructed to do so, removes the Ethernet and IP headers bytes of these packets as instructed by the Card (the Card specifies the number of bytes to be removed from the headers).

## 5.4 Card Operational Modes

The S-CARD operates only in Single-Stream mode (S-Mode). An M-Host MAY support a Card operating in S-Mode or it may reject the S-CARD operating in S-Mode and require that an M-CARD be inserted and that the Card operate in M-Mode only.

The M-CARD physical interface will operate in two modes depending on the capabilities of the Host:

- S-Mode
- M-Mode

The different Card/Host combinations are summarized in Table 5.4–1.

**Table 5.4–1 - Card/Host Combinations and Operating Modes**

	Single-Stream Host	Multi-Stream Host
<b>S-CARD</b>	S-Mode	Host may reject S-CARD*
<b>M-CARD</b>	S-Mode	M-Mode

\* M-Host may optionally support the S-Mode interface which will affect the ability to support multi-stream functionality for that Host, i.e., only a single transport stream may be supported across the interface.

### 5.4.1 S-CARD in S-Mode

S-CARD interface defined in this specification and the corresponding compatible Hosts capable of processing a single transport stream, are built in compliance with the Single-Stream mode (S-Mode) functionality as defined in this document. The S-CARD, when inserted into an M-Host, may or may not be capable of transport stream processing, for both one-way and two-way operation. Said differently, the S-CARD when inserted into a M-Host may not perform in the same manner as if it were inserted into a Host only capable of processing a single transport stream.

### 5.4.2 M-CARD in S-Mode

The M-CARD defined in this specification SHALL function in a single-tuner Host built in compliance with the S-Mode operation as defined in this document. An M-CARD operating in such a Host will be said to be operating in S-Mode.

### 5.4.3 M-CARD in M-Mode

The M-CARD defined in this specification, capable of processing multiple transport streams, SHALL function in a M-Host as defined in this specification document.

The M-CARD physical interface is compatible with the S-CARD physical interface. For M-Mode, the MPEG data flow has been modified to support multiple streams and new APDUs have been added. The command and control interface is a serial interface in M-Mode mode, versus the parallel interface in the S-Mode.

The M-CARD, when inserted into the M-Host, when configured/commanded to do so, could operate in the M-Mode, but only have one stream enabled.

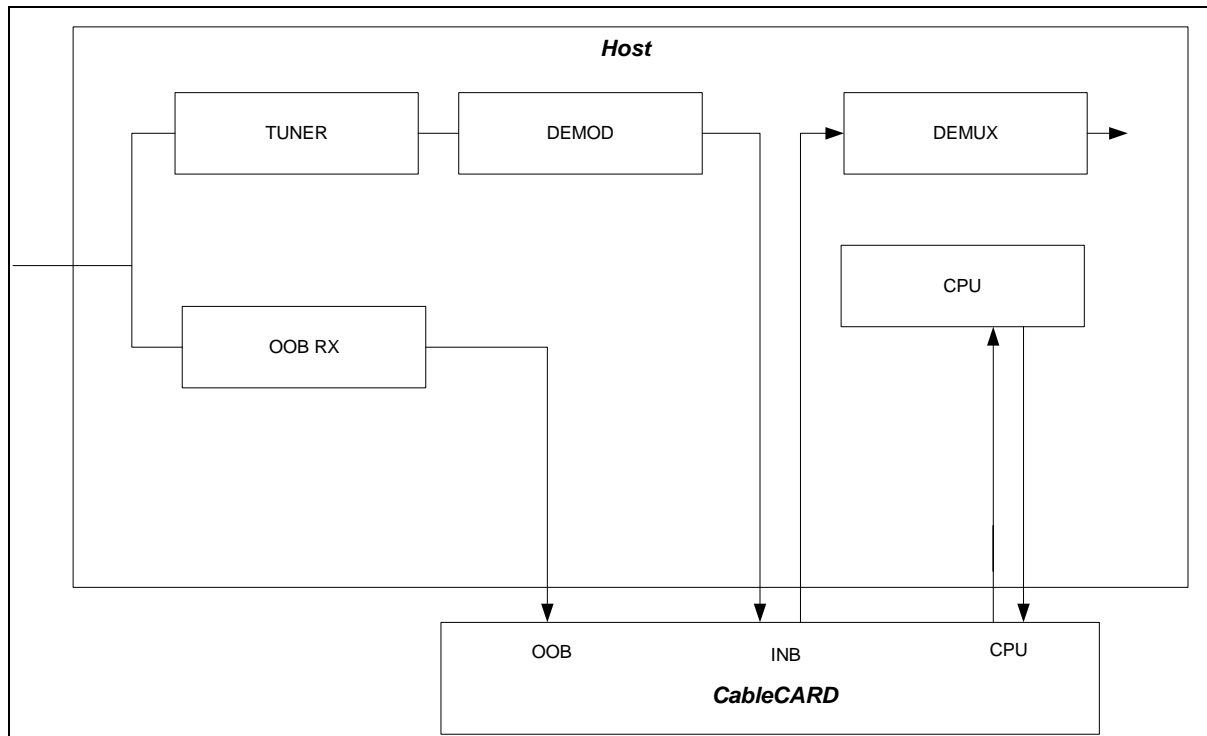
## 5.5 One-way Networks

The configuration shown in Figure 5.5-1 applies where there is a no return channel.

The QPSK transmitter in the Host is not active (and it is therefore omitted from the diagram). The receiver circuit operates in the same manner as described in Section 9.8.

The DSG communication path using the extended channel is intended to provide transport of Out-of-Band messaging over a DOCSIS channel that is traditionally carried on dedicated channels, specifically those defined in

[SCTE55-1] and [SCTE55-2], and is to be capable of supporting a one-way (downstream) transport without requiring return path functionality from the DSG client.



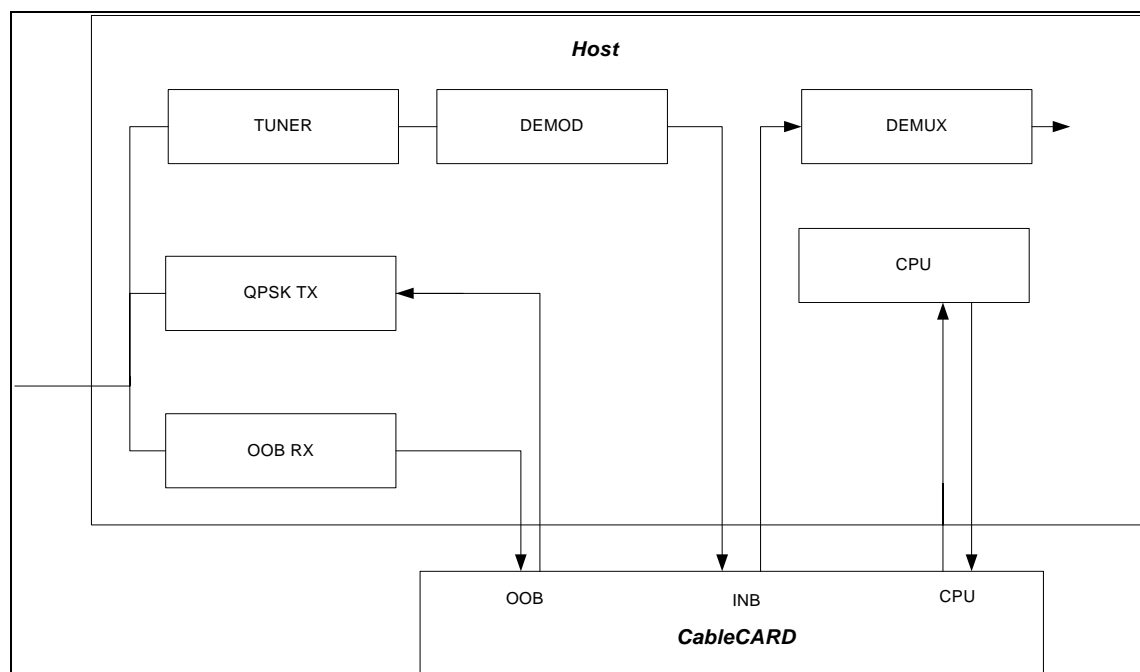
**Figure 5.5-1 - System with One-way Network**

After Card initialization, the Host informs the Card about the available Low Speed Communication resources as defined by Section 9.6. Then, when the Card requires setting up a connection with the cable headend, datagrams are sent to the modem via the CPU interface.

## 5.6 Two-way Networks

Figure 5.6-1 gives a block diagram view of the system when the cable network includes an OOB return Data Channel based on [SCTE55-1] or [SCTE55-2].





**Figure 5.6-1 - System with Two-way Network**

The QPSK receiver circuit in the Host tunes and demodulates the QPSK Forward Data Channel (FDC). The receiver circuit adapts to the 1.544/3.088 Mbps or 2.048 Mbps FDC bit rate, and delivers the bit-stream and clock to the Card. (This data is used primarily to send conditional access entitlement management messages from the cable system to the Card. These messages are beyond the scope of this standard.)

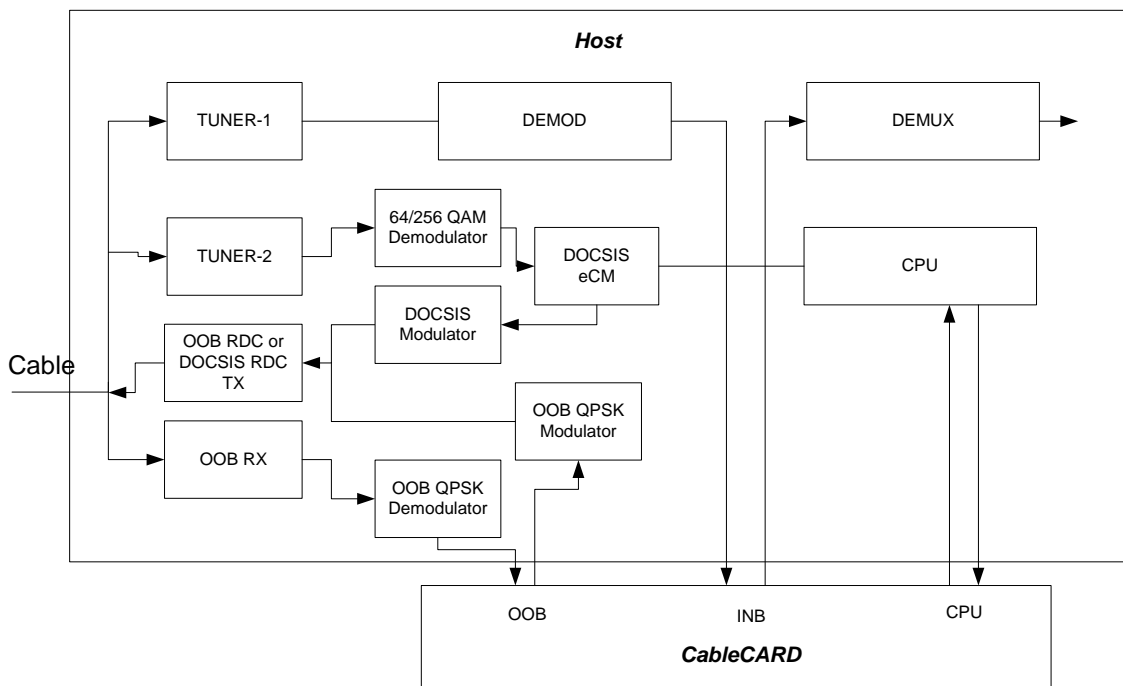
Tuning of the QPSK receiver circuit is under control of the Card, as explained in Section 9.8. The tuning range is between 70 and 130 MHz.

In the return path, the Card generates QPSK symbols and clock and transfers them to the QPSK transmitter circuit in the Host. The transmitter circuit adapts to the 1.544/3.088 Mbps or 0.256 Mbps RDC bit rate. The QPSK transmitter circuit modulates the QPSK symbols onto a narrow band carrier.

Tuning and level control of the QPSK transmitter are under control of the Card as explained in Section 9.8. The tuning range is between 5 MHz and 42 MHz.

## 5.7 Two-way Networks with DOCSIS

The configuration shown in Figure 5.7-1 applies where DSG capability via a DOCSIS modem exists in the Host.



**Figure 5.7-1 - System with DOCSIS Two-way Network**

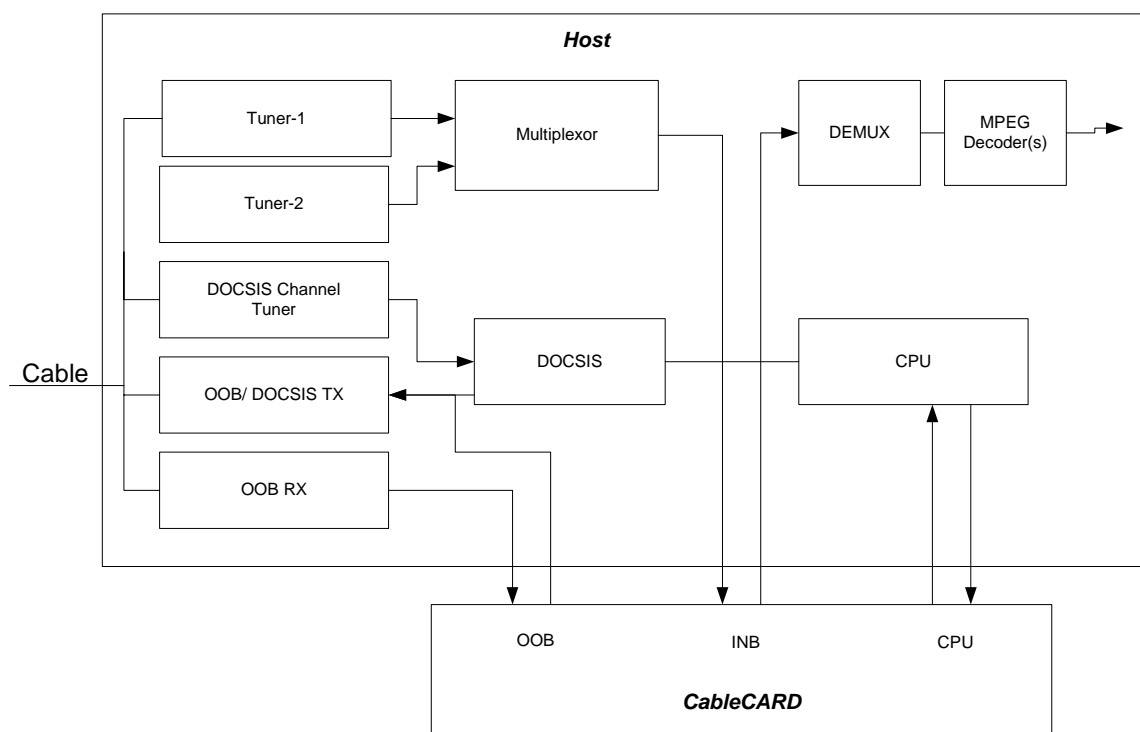
In this configuration a single upstream transmit path is shared between the Card and the DOCSIS modem. In order to prevent conflict between the DOCSIS upstream and the OOB RDC, the system will operate in one of two modes.

- **OOB mode** – The downstream Conditional Access Messages and network management messages will be delivered to the Card via the QPSK receive interface on the Card using, e.g., [SCTE55-1], [SCTE55-2], or other agreed OOB specification. The upstream Conditional Access Messages and network management messages will be transmitted from the Card via the QPSK transmit interface on the Card using, e.g., [SCTE55-1], [SCTE55-2].
- **DSG mode** – The downstream Conditional Access Messages and network management messages will be delivered to the Card by the Extended Channel using the DSG Service type in the DOCSIS downstream in accordance with the DOCSIS Set-top Gateway Specification [DSG]. The upstream Conditional Access Messages and network management messages will be transmitted from the Card via IP over the DOCSIS upstream channel using the Extended Channel. The DOCSIS bi-directional channel can be used by any applications running in the Host, simultaneously with the Card's communication with the headend via the Extended Channel using DOCSIS. The use of the Extended Channel by the Card for IP flows does not change DSG usage of the DSG Service type on the Extended Channel.

The mode used is based on whether the DOCSIS Set-top Gateway is supported by the network. The Card informs the Host which of these modes is to be used as detailed later in this specification.

## 5.8 M-CARD Device Functional Description

The functional elements of a Host and M-CARD are shown in the following figure. Shown in the diagram is an example of a possible M-Mode implementation that includes multiple FAT tuners and an eCM.



**Figure 5.8-1 - Host and M-CARD Device Block Diagram Example**

The multiplexer sends one complete MPEG transport packet at a time across the MPEG interface to the M-CARD. The Host identifies each of the transport stream packets in the multiplex by appending a pre-header to the MPEG-2 transport header. This pre-header contains an 8-bit transport stream ID value, indicating to which transport stream the appended packet belongs.

In order to support multiple streams, the bandwidth of the physical interface between the Card and the Host device is increased from 27/38Mbps to support up to 200Mbps of aggregate MPEG transport packet data both into and out of the Card simultaneously. This is sufficient for five simultaneous transport streams from up to five 256QAM tuners/demodulators, or up to six streams from six 64QAM tuners/demodulators.

For the Card, the maximum number of unique MPEG transport streams input into the Card from the Host will be greater than three. The maximum number of programs that the Card's CA system can request that the Card simultaneously decrypt can be greater than or equal to four. In addition, the maximum number of elementary streams that the Card can manage can be greater than or equal to sixteen (16). The Card MAY request a maximum of 8 PIDs be transmitted to it. Section 9.12, M-Mode Device Capability Discovery, identifies this functionality.

In addition to increasing the number of transport streams that can be transmitted across the interface, this specification also adds capability to the Card-Host command interface to enable the decryption of multiple programs within each transport stream. Therefore, it is possible for the Host to request that multiple programs from each transport stream be decrypted. Through the CA\_PMT structure, which is passed from the Host to the Card, the Host indicates which programs and which individual elementary streams within each transport stream are to be decrypted by the Card. Conversely, upon initialization, the Card indicates to the Host the number of simultaneous programs and simultaneous elementary streams (PIDs) it can decrypt. The Host can, therefore, keep track of how many of each of these resources it has used to determine how many additional programs and elementary streams it can request to be decoded. These commands are diagrammed in the following table.

**Table 5.8–1 - Card-Host Resource Communication**

Host		Card
	←	Maximum number of transport streams supported
	←	Maximum number of simultaneous programs supported
	←	Maximum number of simultaneous elementary streams (PIDs) supported
CA_PMT structure(s) (one per program)	→	

An example of the resources that need to be tracked are shown in Table 5.8–2, where the Host has requested six programs consisting of two movies, one multi-angle movie, and three music programs, for a total of six programs, eleven elementary streams, and three transport streams. If the Card had indicated that it could support 4 transport streams, 16 programs, and 32 elementary streams, the Host knows that there are sufficient resources for additional decryption requests. If the Card only supported simultaneous decryption of a maximum of six programs, the Host would know not to send any additional program decrypt requests. If the Card indicated that it could only support three simultaneous streams, the Host could optionally re-multiplex two separate transport streams into one and pass it to the Card instead, as long as the decryption request remained under the maximum limits indicated by the Card.

**Table 5.8–2 - Resource Example Request**  
**From the Host to Decrypt 11 Elementary Streams From 6 Programs Across 3 Transport Streams**

Transport Layer	Program Layer	Elementary Layer
Transport Stream 1	Program 1	Video ES
		Audio ES
	Program 2	Video ES
		Audio ES
Transport Stream 2	Program 3	Video ES 1
		Video ES 2
		Audio ES 1
		Audio ES 2
Transport Stream 3	Program 4	Audio ES
	Program 5	Audio ES
	Program 6	Audio ES

## 5.9 Inband Interface - MPEG Data Flow

In S-Mode, the CHI SHALL support the transport stream interface data rates of 26.97035 Mb/s and 38.81070 Mb/s averaged over the period between the sync bytes of successive transport packets with allowable jitter of +/- one MCLKI clock period.

The Card's MPEG data flow uses two separate 8-bit buses, one for input, and one for output for the MPEG data. All of the Host MPEG data is transmitted through the Card. The Card only descrambles the selected program indicated by the Host. If the program is marked as high value content, non-zero EMI, that program is scrambled across the CHI.

In M-Mode only, packets from multiple transport streams are temporally multiplexed and sent across the parallel MPEG transport interface. In addition, a header is added before each packet for identification. The clock rate of the interface is increased in order to support the increased number of packets.

## 5.10 OOB Interface

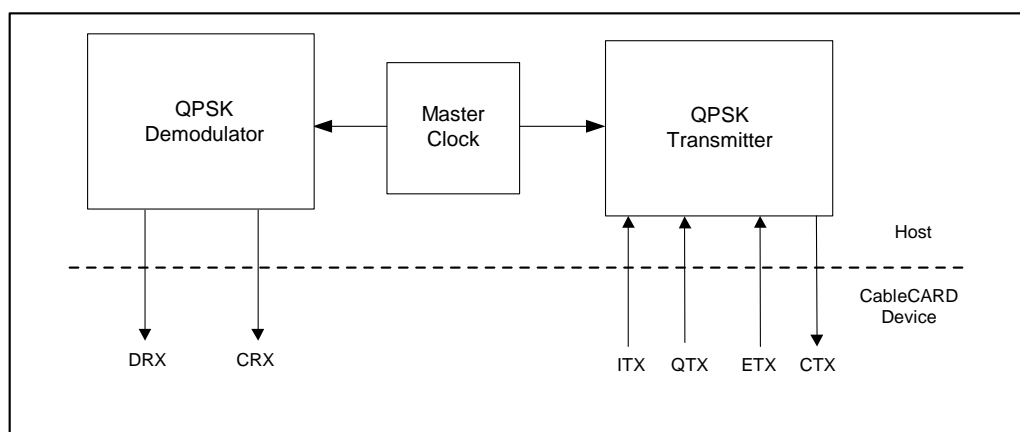
The S-CARD device was modified from the [NRSSB]-defined Card to include signaling for the ANSI/SCTE 55-1 and 55-2 OOB methods operation. A later modification added the DSG functionality where the OOB data can be transmitted over the extended channel interface to the Card.

The Host provides the QPSK physical layer to support OOB (FDC and RDC) communications according to [SCTE55-1] and [SCTE55-2]. The data link and media access control protocols for [SCTE55-1] and [SCTE55-2] are implemented in the Card. See Figure 5.10-1 below.

The interface data rates are:

- Forward Receiver: 1.544/3.088 Mbps and 2.048 Mbps
- Reverse Transmitter: 772/1544 ksymbol/s and 128 ksymbol/s (i.e., 1.544/3.088 Mbps and 256 kbps)

The transmit and receive interfaces for the Card OOB Interface are shown in Figure 5.10-1 below. The receiver interface comprises a serial bit stream and a clock, while the transmitter interface comprises I and Q data, a symbol clock, and a transmit-enable signal. The clock signal should be transferred from the Host to the Card, as shown in Figure 5.10-1.

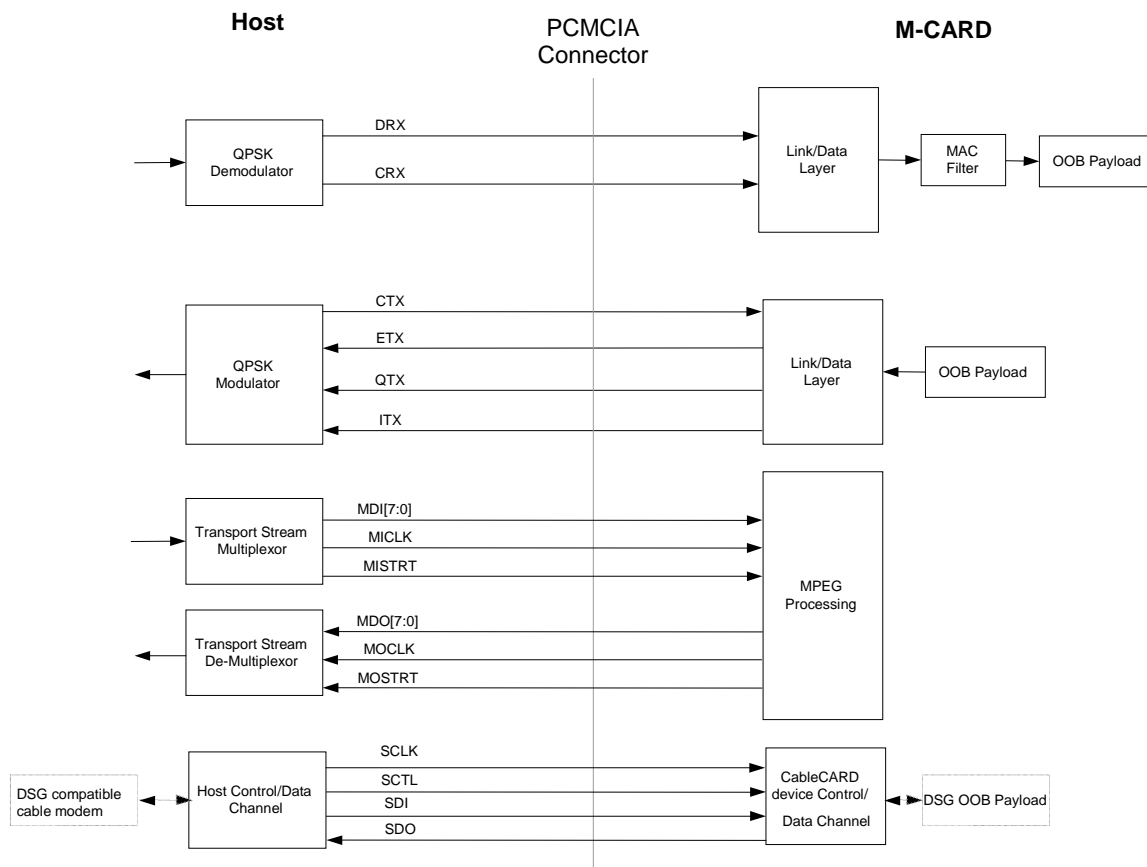


**Figure 5.10-1 - CableCARD Out-of-Band Interface**

The Card operating in S-Mode or M-Mode SHALL support DSG, ANSI/SCTE 55-1 and ANSI/SCTE 55-2 Out-Of-Band (OOB) methods defined in [DSG] [SCTE55-1] and [SCTE55-2] respectively. Only one method will be active at a time.

For the DSG operation, the Host SHALL support DOCSIS as defined in the [OCHD2] specification. Operation will follow the [DSG] specification.

The following diagram shows the connections for the MPEG transport and Out-Of-Band (OOB) data flows for the Card operating in M-Mode:



**Figure 5.10-2 - M-Mode: CHI Diagram**

### 5.10.1 QPSK

The common modulation method for ANSI/SCTE 55-1 and ANSI/SCTE 55-2 is QPSK. This allows the Host to incorporate a common receiver and transmitter for support of the legacy QPSK signaling. The receive signals (data and clock) are passed to the Card, which performs all the necessary MAC and higher layers of operation.

### 5.10.2 DSG

The Host performs all the DOCSIS operations and through the use of DSG tunnels, allows for the transmission of the OOB data to the Card.

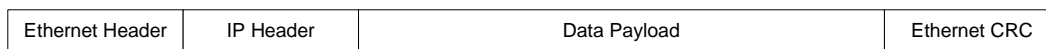
The Host DOCSIS cable modem provides the physical data link and media access control protocols. Unlike the OOB mode, the data link and media access control protocols for ANSI/SCTE 55-1 and ANSI/SCTE 55-2 are bypassed in the Card. The downstream communications are implemented in accordance with the DOCSIS Set-top Gateway Specification [DSG]. The upstream Conditional Access Messages and network management messages will be transmitted from the Card via IP over the DOCSIS upstream channel using the Extended Channel.

The interface data rates are:

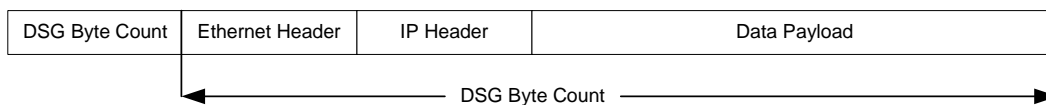
- Downstream direction: 2.048 Mbps

The first two bytes of the frame are the total number of bytes following in the frame, i.e., they do not include this two-byte length field. There is no CRC check required on the frame, as the interface between the Host and Card is reliable. It is the responsibility of the Card vendor to implement error detection in the DSG encapsulated data. The Card should disregard any invalid packets received from the Host. The Host SHALL provide buffer space for a minimum of two DSG PDUs, one for transmission to the Card and one for receiving from the DOCSIS channel. Informational note: The DSG rate limits the aggregate data rate to 2.048 Mbps to avoid buffer overflow.

Figure 5.10-3 below shows how the DSG packets are transported across the Card interface with and without removal of the IP header bytes. The CRC is removed, then when requested, the Ethernet and IP header is removed, and a two-byte field containing the byte count of the resulting data is transmitted across the Card interface.



DSG Packet From Embedded DOCSIS CM



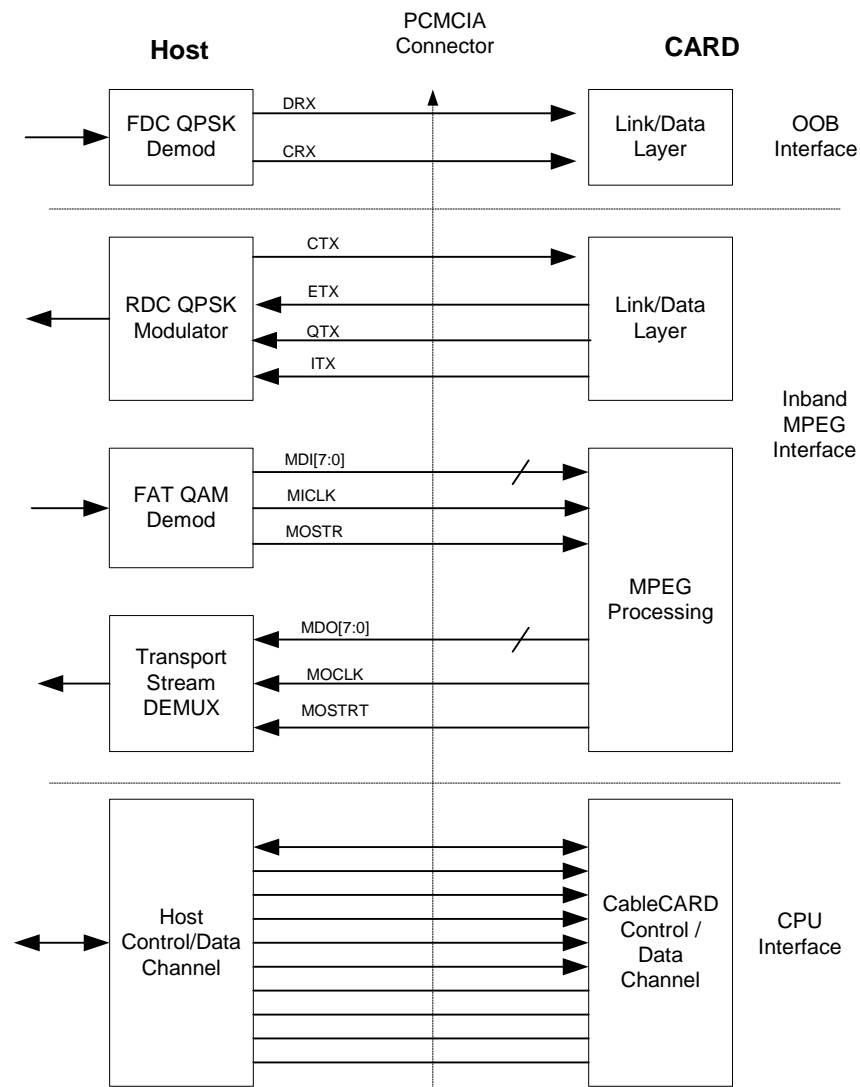
DSG Packet Across CableCARD Device Interface (Remove\_Header\_Bytes = 0)



DSG Packet Across CableCARD Device /Host Interface (Remove\_Header\_Bytes = Ethernet Header + IP Header Size)

**Figure 5.10-3 - DSG Packet Format Across Card Interface**

The following diagram shows the connections for the MPEG transport and Out-Of-Band (OOB) data flows for the Card operating in S-Mode:



**Figure 5.10-4 - S-Mode CHI Diagram**



## **6 DELETED**

NOTE: This section intentionally left blank to retain document's section numbering.

## 7 PHYSICAL INTERFACE

The Card will use the PCMCIA Cardbus Type II physical form factor. The electrical interface differs between S-Mode and M-Mode. The M-Host senses the presence of an M-CARD and operates as defined below. If the M-Host detects that the Card is not an M-CARD, it can reject the Card.

The M-Host MAY reject an S-CARD. The M-CARD will start up directly in M-Mode when VPP1# is logic low, and VPP2# is powered. When VPP1# and VPP2# is powered, the M-CARD will follow the PC Card start-up procedure for personality change to S-Mode.

The S-CARD will always follow the PC Card start-up procedure for personality change to S-Mode.

### 7.1 Electrical Characteristics

When in memory card mode, just after reset, the pin assignments in the left hand column of Tables 4-1 and 4-2 of [PCMCIA2] are used. When the module is configured as the Common Interface variant during the initialization process, the following reassignments are made: The pins carrying signals A15-A25, D8-D15, BVD1, BVD2, and VS2# are used to provide high-speed input and output buses for the MPEG-2 multiplex data. IOIS16# is never asserted and CE2# is ignored. All other pins retain their assignment as an I/O & Memory Card interface.

The following are the different pin assignments for the Card interface for the different modes.

**Table 7.1-1 - Card Interface Pin Assignments**

Pin #	EIA-679B part B		PC Card Mode		S-Mode		M-Mode	
	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output
1	GND	DC gnd	GND	DC gnd	GND	DC gnd	GND	DC gnd
2	D3	I/O	D3	I/O	D3	I/O		
3	D4	I/O	D4	I/O	D4	I/O		
4	D5	I/O	D5	I/O	D5	I/O		
5	D6	I/O	D6	I/O	D6	I/O		
6	D7	I/O	D7	I/O	D7	I/O		
7	CE1#	I	CE1#	I	CE1#	I		
8	A10	I	A10	I				
9	OE#	I	OE#	I	OE#	I		
10	A11	I	A11	I				
11	A9	I	A9	I	DRX	I	DRX	I
12	A8	I	A8	I	CRX	I	CRX	I
13	A13	I	A13	I			MOCLK	O
14	A14	I	A14	I	MCLKO	O		
15	WE#	I	WE#	I	WE#	I		
16	IREQ#	O	READY	O	IREQ#	O		
17	VCC		VCC	DC in	VCC	DC in	VCC	DC in
18	VPP1		VPP1	DC in	VPP1	DC in	VPP1	I
19	MIVAL	I	A16	I	MIVAL	I		
20	MCLKI	I	A15	I	MCLKI	I		
21	A12	I	A12	I			MICLK	I
22	A7	I	A7	I	QTX	O	QTX	O
23	A6	I	A6	I	ETX	O	ETX	O
24	A5	I	A5	I	ITX	O	ITX	O
25	A4	I	A4	I	CTX	I	CTX	I
26	A3	I	A3	I				
27	A2	I	A2	I			SCTL	I

Pin #	EIA-679B part B		PC Card Mode		S-Mode		M-Mode	
	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output	Signal and Pin Name	Card Input or Output
28	A1	I	A1	I	A1		SCLK	I
29	A0	I	A0	I	A0		SDI	I
30	D0	I/O	D0	I/O	D0	I/O		
31	D1	I/O	D1	I/O	D1	I/O		
32	D2	I/O	D2	I/O	D2	I/O		
33	IOIS16#		WP	O	IOIS16#	O	MDET	O
34	GND		GND	DC	GND	DC	GND	DC
35	GND		GND	DC	GND	DC	GND	DC
36	CD1#		CD1#	O	CD1#	O	CD1#	O
37	MDO3	O	D11	I/O	MDO3	O	MDO3	O
38	MDO4	O	D12	I/O	MDO4	O	MDO4	O
39	MDO5	O	D13	I/O	MDO5	O	MDO5	O
40	MDO6	O	D14	I/O	MDO6	O	MDO6	O
41	MDO7	O	D15	I/O	MDO7	O	MDO7	O
42	CE2#	I	CE2#	I	CE2#	I		
43	VS1#	O	VS1#	O	VS1#	O	VS1#	O
44	IORD#	I	RFU		IORD#	I		
45	IOWR#	I	RFU		IOWR#	I		
46	MISTRT	I	A17	I	MISTRT	I	MISTRT	I
47	MDI0	I	A18	I	MDI0	I	MDI0	I
48	MDI1	I	A19	I	MDI1	I	MDI1	I
49	MDI2	I	A20	I	MDI2	I	MDI2	I
50	MDI3	I	A21	I	MDI3	I	MDI3	I
51	VCC		VCC	DC in	VCC	DC in	VCC	DC in
52	VPP2		VPP2	DC in	VPP2	DC in	VPP2	DC in
53	MDI4	I	A22	I	MDI4	I	MDI4	I
54	MDI5	I	A23	I	MDI5	I	MDI5	I
55	MDI6	I	A24	I	MDI6	I	MDI6	I
56	MDI7	I	A25	I	MDI7	I	MDI7	I
57	MCKLO	I	VS2#	O	VS2#	O	VS2#	I/O
58	RESET	O	RESET	I	RESET	I	RESET	
59	WAIT#	O	WAIT#	O	WAIT#	O		
60	INPACK#	O	RFU		INPACK#	O	SDO	O
61	REG#	I	REG#	I	REG#	I		
62	MOVAL	O	BVD2	O	MOVAL	O		
63	MOSTRT	O	BVD1	O	MOSTRT	O	MOSTRT	O
64	MDO0	O	D8	I/O	MDO0	O	MDO0	O
65	MDO1	O	D9	I/O	MDO1	O	MDO1	O
66	MDO2	O	D10	I/O	MDO2	O	MDO2	O
67	CD2#	O	CD2#	O	CD2#	O	CD2#	O
68	GND		GND	DC	GND	DC	GND	ground

“I” indicates signal is input to Card; “O” indicates signal is output from Card.

Blank = Unused pin

It is recommended that at least 12 bits of address be decoded on the Card (4096 bytes) during memory accesses. A lower number of bits, as specified in the CIS, SHALL be decoded during I/O accesses as defined in section 4.3 of [PCMCIA2].

## 7.2 S-Mode Start-Up

The Card starts up as a PC Card, then goes through a personality change according to the PC Card standard.

### 7.2.1 Card Port Custom Interface (0x341)

The S-CARD interface is registered to the PC Card Standard as the “POD Module Custom Interface” with the interface ID number (STCI\_IFN) allocated to equal hexadecimal 341 (0x341). In case the Host is not capable of operating with the Card, the Host SHALL ignore the Card.

The Card SHALL present the 16-bit PC Card memory-only interface following the application of VCC or the RESET signal. When operating in this configuration, D7-D0 are retained as a byte-oriented I/O port, and the capability to read the Attribute Memory is retained.

Only two address lines are required for four Bytes of register space. In S-Mode, pin CE2# is assigned to select the Extended Channel function required for the Card CPU interface to enable the access to the Extended Channel resource. Pin IOIS16# is never asserted.

In S-Mode the CHI SHALL be required to support transport stream interface data rates of 26.97035 Mb/s and 38.81070 Mb/s averaged over the period between the sync bytes of successive transport packets with allowable jitter of +/- one MCLKI clock period.

## 7.3 Interface Functional Description

### 7.3.1 S-Mode

Table 7.1–1 shows the function of various PC Card signals when the Card is operating in S-Mode and the Port custom interface mode is set to active in the Host.

Differences between EIA-679-B Part B and the Card when running in S-Mode are identified in Table 7.1–1. The CHI Specifications affect the A4 to A9 signals, which are now assigned to the OOB RF I/Os, and CE2#, which is used to access the Extended Channel. The MCLKO is provided on pin 14 to be fully PC Card compliant. This is a modification from EIA-679-B (Part B). Pin 57 remains the PC Card VS2# signal. Table 7.1–1 indicates the differences between EIA 679-B Part B and this document, namely pins 11, 12, 14, 22, 23, 24, 25, 42 and 57.

### 7.3.2 M-Mode

When operating in M-Mode, the M-CARD SHALL have the connector pin assignment as described in Table 7.1–1. Note that the EIA 679-B Part B, the PCMCIA PC Card Mode, S-Mode and M-Mode pin assignments are included for reference.

### 7.3.3 Card Signal Descriptions

#### 7.3.3.1 Power

<b>VCC</b>	If the Card is interfacing to a Host that supports the S-Mode, these pins are power pins that initially supply 3.3V per Section 7.4.1.1. If the Card is interfacing to a Host using M-Mode, the VCC pins are at a High-Z until Card-type identification and discovery is performed. After identification of the M-Card that will be operating in M-Mode is detected, these pins are powered up to 3.3V.
<b>GND</b>	As defined in section 2.1.1 of [PCMCIA2].
<b>VPP1</b>	If the Card is interfacing to a Host that supports the S-Mode, this pin is a power pin that initially supplies 3.3V and can be switched to 5V per Section 7.4.1.1. If the Card is interfacing to a Host using M-Mode, this pin is at a High-Z until Card-type identification and discovery is performed. The VPP1 pin is then set to logic low to indicate that the Card will be interfacing to the Host in M-Mode.

**VPP2** If the Card is interfacing to a Host that supports the S-Mode, this pin is a power pin that initially supplies 3.3V and can be switched to 5V per Section 7.4.1.1. If the Card is interfacing to a Host using M-Mode, this pin is at a High-Z until Card-type identification and discovery is performed. The VPP2 pin is then configured to a 5V supply pin.

### 7.3.3.2 Sense

**CD2::1#** As defined in section 2.2.1 of [PCMCIA2].

**VS2::1#** As defined in section 2.2.2 of [PCMCIA2]. For M-Mode only, 3.3V will be supported (VS1# = GND, VS2# = High-Z). VS2# is also used during Card-type detection and tied directly to MDET.

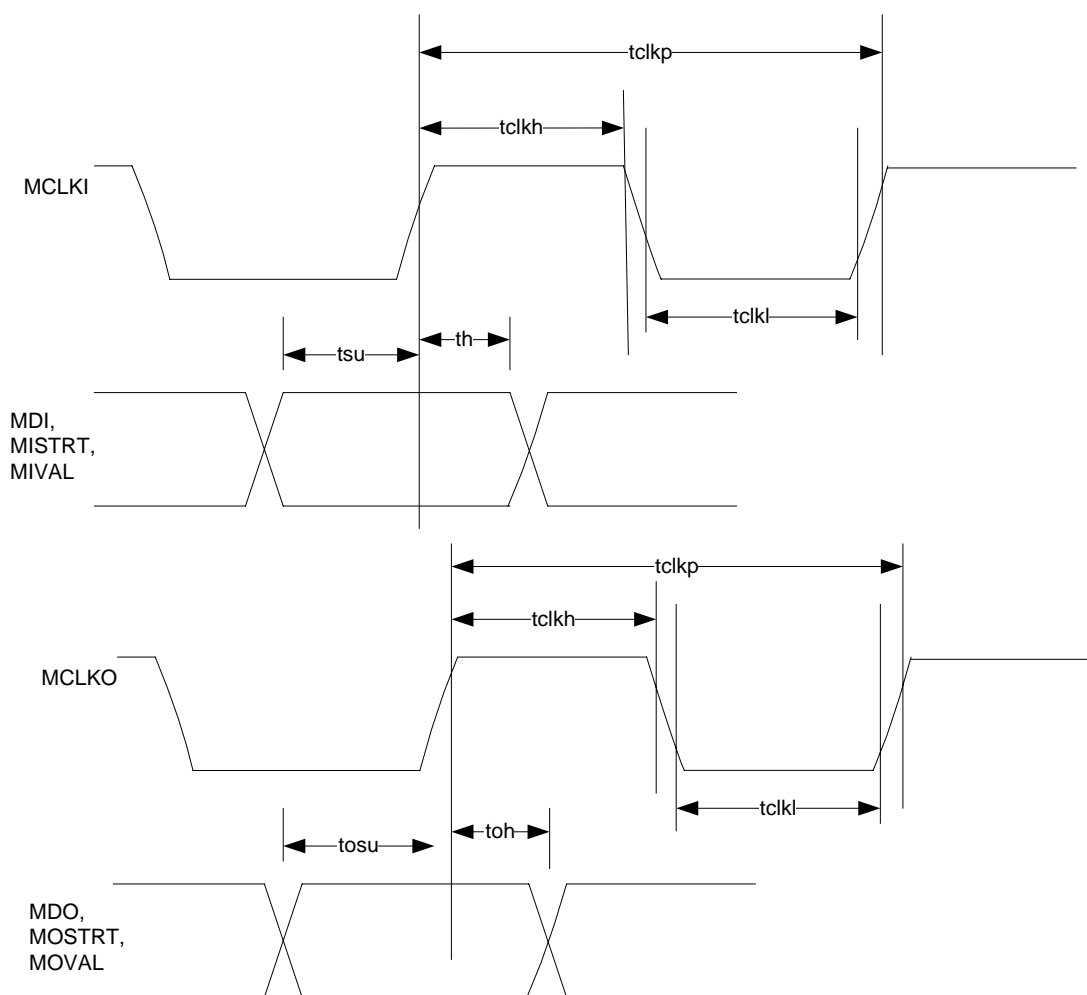
**MDET/IOIS16#** M-Card detect signal used by the Host to identify if Card will be operating in S-Mode or M-Mode. For M-Mode, MDET is tied to VS2#. For S-Mode, IOIS16# is not tied to VS2#.

### 7.3.3.3 PCMCIA Signals

As defined in sections 4.4, 4.6, and 4.7 of [PCMCIA2] with the following additions:

For S-Mode, MPEG-2 transport stream interface input and output buses are provided (MDI0-7, MDO0-7). Control signals MCLKI, MCLKO, MISTRT, MIVAL, MOSTRT, MOVAL are also provided. MCLKI runs at the rate at which bytes are offered to the Card on MDI0-7. MCLKO runs at the rate at which bytes are offered by the Card on MDO0-7. For Cards which pass the transport stream through, then MCLKO will in most cases be a buffered version of MCLKI with a small delay. For Cards which originate data, for example a detachable front-end or a network connection, then MCLKO may be derived from that data source. Figure 7.3-1 shows the relative timing relationship of the data signals associated with the MPEG-2 transport stream interface and MCLKI, MCLKO, and Table 7.3-1 gives limits to these timing relationships. Note that the specification for output timing limits can normally be met easily by generating the output from the falling edge of MCLKO. There is no specification for delay between MCLKI and MCLKO. In the case of a Card providing its own MCLKO, they may not even be the same frequency. Both MCLKI and MCLKO SHALL be continuous. It is not intended that burst clocking should be employed. Bursty data is handled by the appropriate use of MIVAL and MOVAL.

MISTRT is valid during the first byte of each transport packet on MDI0-7. The edge timing of this signal SHALL be the same as for MDI0-7. See Figure 7.3-1.



**Figure 7.3-1 - Timing Relationships for Transport Stream Interface Signals**

**Table 7.3-1 - Timing Relationship Limits**

Item	Symbol	Min	Max
Clock Period	Tclkp	110 ns	
Clock High time	Tclkh	40 ns	
Clock Low time	Tclkl	40 ns	
Input Data Setup	Tsu	15 ns	
Input Data Hold	Th	10 ns	
Output Data Setup	Tosu	20 ns	
Output Data Hold	Toh	15 Ns	

MIVAL indicates valid data bytes on MDI0-7. All bytes of a transport packet may be consecutive in time, in which case MIVAL will be at logic 1 for the whole of the duration of the transport packet. However certain clocking strategies adopted in Hosts may require there to be gaps of one or more byte times between some consecutive bytes within and/or between transport packets. In this case MIVAL will go to logic 0 for one or more byte times to indicate data bytes which should be ignored.

MDO0-7 is the output bus for the MPEG-2 transport stream interface. Where MCLKO is derived from MCLKI, and correspondingly the data on MDO0-7 is a delayed and possibly descrambled version of the data on MDI0-7, then the timing relationship between the input data and the output data SHALL be governed by the rules in Section 7.3.6.1 of the document.

MOSTRT is valid during the first byte of an output transport packet.

MOVAL indicates the validity of bytes on MDO0-7 in a similar manner to MIVAL. MOVAL may not necessarily be a time-delayed version of MIVAL, see Section 7.3.6.1.

Support for Interrupt Requests by Hosts as defined in section 4.4.7 of [PCMCIA2] and 7.6.1.1 of this document. Whenever the Card responds to an I/O operation it SHALL assert INPACK#, see section 4.4.22 of [PCMCIA2].

### 7.3.3.4 Card Device Signals

<b>DRX</b>	QPSK receive data input to the Card from the Host.
<b>CRX</b>	QPSK receive gapped clock input to the Card from the Host in which some of the clock cycles are missing, creating an artificial gap in the clock pattern.
<b>ITX</b>	QPSK transmit I-signal output from the Card to the Host and are represented directly to the phase states as defined in [SCTE55-2].
<b>QTX</b>	QPSK transmit Q-signal output from the Card to the Host and are represented directly to the phase states as defined in [SCTE55-2].
<b>ETX</b>	QPSK transmit enable output from the Card to the Host. It is defined to be active high.
<b>CTX</b>	QPSK transmit gapped symbol clock input to the Card from the Host.
<b>MCLKI</b>	MPEG transport stream clock from Host to the Card operating in S-Mode.
<b>MICLK</b>	MPEG transport stream clock from the Host to the Card operating in M-Mode.  <b>Note:</b> When the M-CARD is operating in S-Mode, MICLK (A12 pin 21) will have the characteristics of CCLK signal type. Additional details in Section 7.4.1.3.2.
<b>MISTRT</b>	MPEG transport stream input packet start indicator from the Host to the Card operating in S-Mode. The Card operating in M-Mode, used to indicate the start of a CableCARD MPEG Packet, CMP. It is asserted at the same time as the first byte of the CMP header.
<b>MDI[7:0]</b>	An 8-bit wide MPEG transport stream input data bus from the Host to the Card.
<b>MCLKO</b>	MPEG transport stream clock from the Card to the Host operating in S-Mode.
<b>MOCLK</b>	MPEG transport stream clock from the Card to the Host operating in M-Mode. The MOSTRT and MDO[7:0] signals SHALL be clocked into the Host on the rising edge of MOCLK. This signal SHALL be derived from MICLK and SHOULD operate at 27 MHz.
<b>MOSTRT</b>	MPEG transport stream output packet start indicator from the Card operating in S-Mode to the Host. The Card, operating in M-Mode, is used to indicate the start of a CMP. It is asserted at the same time as the first byte of the CMP header.
<b>MDO[7:0]</b>	An 8-bit MPEG transport stream output data bus from the Card to the Host.
<b>SCLK</b>	CPU interface serial clock from the Host to the Card operating in M-Mode. The clock is a continuously running clock (not gapped) and has a nominal frequency of 6.75 MHz.
<b>SCTL</b>	CPU interface serial interface control signal from the Host into the Card operating in M-Mode. It signals the start of a byte of data and the beginning of a packet being transferred across the interface.
<b>SDI</b>	CPU interface serial data from the Host into the Card operating in M-Mode.
<b>SDO</b>	CPU interface serial data from the Card operating in M-Mode out to the Host.

**RESET** Reset input to the Card. RESET is active high or asserted when a logic high.

**Table 7.3–2 - Transmission Signals**

Signal	Rates	Type
DRX	1.544/3.088 and 2.048 Mbps	I
CRX	1.544/3.088 or 2.048 MHz	I
ITX	772/1544 and 128 Ksymbol/s	O
QTX	772/1544 and 128 Ksymbol/s	O
ETX	TX Enable	O
CTX	772/1544 and 128 KHz	I

When ETX is inactive, the values of ITX and QTX are not valid and the upstream transmitter SHALL NOT transmit such values. When ETX is active, the values of ITX and QTX are both valid and the upstream transmitter SHALL transmit these values.

The MICK signal SHALL always operate at 27Mhz.

The MISTRT and MDI[7:0] data signals SHALL be clocked into the Card on the rising edge of MICK.

The MOSTRT and MDO[7:0] signals SHALL be clocked into the Host on the rising edge of MOCLK. This signal SHALL be derived from MICK and MICK SHALL operate at 27 MHz.

### 7.3.4 Card Type Identification

To support 3.3 Volts, the Host SHALL use the detection mechanisms described in section 3 of [PCMCIA2]. To determine Card operating voltage, Hosts SHALL follow the rules for type of socket according to whether they provide 3.3v working - section 3.2 of [PCMCIA2]. Hosts need not support the detection mechanisms for CardBus PC Cards, but may optionally do so - section 3.3 of [PCMCIA2].

To allow Hosts to work with the Card operating in M-Mode, the physical interface, the PC Card keying mechanisms, and the initialization steps for the Hosts SHALL be the same as those defined for the Card operating in S-Mode and the S-CARD references in this document. Thus, a step is added to the existing initialization steps that allows M-Host to reject Cards operating in S-Mode and allows the M-Card to determine what type of Host configuration to support.

#### 7.3.4.1 S-Mode

The Host has two different ways to recognize a Card operating in S-Mode, using the Application Info EIA-679-B Part B resource, or at the physical level as defined by PCMCIA.

In PCMCIA memory mode, the Host accesses the Card's Attribute Memory to read the Card Information Structure (CIS) on the even addresses (first byte at address 0x 000, second byte at address 0x 002, etc.). Since the Card interface is a PC Card Custom Interface, the CIS SHALL include a custom interface subtuple (CCST\_CIF) that provides the interface ID number (STCI\_IFN) defined by PCMCIA (0x341).

For a more explicit identification, the CIS also includes in the tuple CISTPL\_VER\_1, the field name of the product of subtuple TPLLV1\_INFO defined as "OPENCABLE\_POD\_MODULE".

This information in the CIS is mandatory to ensure backup operation in case of trouble when the CI stack is lost (e.g., power shut down, Card extraction).

Note: for the Card operating in S-Mode, PIN 33, ISIO16# should not be tied to VS2# through the Pull-up resistor to 3.3V.



### 7.3.4.1.1 Memory Function

The Host SHALL support Attribute Memory function described in Section 4.6 of [PCMCIA2]. Attribute Memory function support by Hosts is mandatory. Note that Attribute Memory is byte-wide - Attribute Memory data only appears on data lines D7-D0. Also consecutive bytes are at consecutive even addresses (0, 2, 4, etc.). Note also that Attribute Memory must still be able to be read or written to even when the Card is configured to operate with the Common Interface. Common Memory function support in the Host is optional. Cards SHALL NOT use Common Memory.

### 7.3.4.1.2 Timing Function

The Host SHALL support Attribute Memory Timing functions described in Section 4.7 of [PCMCIA2]. Attribute Memory support by Hosts is mandatory. Common Memory support in the Host is optional.

### 7.3.4.2 M-Mode

The M-Host SHALL determine whether the Card is an S-CARD or an M-CARD before applying power to the VCC and VPP pins on the CHI. The Card SHALL be designed such that when un-powered, VS1# SHALL = ground and VS2# SHALL be connected directly to MDET. Upon removal/insertion of the Card, when operating in M-Mode, the Host SHALL use the CD1# and CD2# pins to determine when to power up/down the VCC and VPP interface pins. M-Hosts SHALL:

1. Set the state of VPP1= Logic Low (grounded or pulled-down) and VPP2= open circuit.
2. Pull-up CD1#, CD2#, VS1#, VS2# and MDET to 3.3VDC using a pull-up resistor as defined in table 4-19 of [PCMCIA2].
3. Detect the presence of a Card using CD1# and CD2# as defined in section 4.10 of [PCMCIA2].
4. High Z VCC pins until after the CableCARD device type is determined.
5. Determine if VS2# is tied to MDET. The M-Host MAY test this by toggling VS2# and watching to see that MDET tracks VS2#. If VS2# is tied to MDET, the M-Host SHALL proceed with the power up in M-Mode; otherwise, if the M-Host does not support S-Mode on the CHI the Host SHALL NOT power the PC Card and SHALL display an error message indicating the Card inserted is not supported as defined in Annex B of this document.
6. Apply power to the VCC pins (3.3V) and VPP2 (5.0V) once the Host determines that the Card is an M-CARD. VPP2 is provided to support an M-CARD that contains an optional smart card.

The M-Host will not be required to support PC Card memory-only interface; it MAY immediately operate in the M-Mode.

As power is applied to the M-CARD, all interface pins on the M-CARD SHALL be defined in a manner such that it will not contend with the Host until it knows what mode to operate in. The operating mode is determined by the Card detecting the logic levels of the VPP1 and VPP2 pins when RESET is de-asserted. If the VPP1 and VPP2 pins are at a logic high then the Card SHALL initialize to S-Mode. If the VPP1 pin is at a logic low and the VPP2 pin is a logic high (5V), the Card SHALL initialize to M-Mode.

If the VPP1/VPP2 configuration is one of the two reserved settings in Table 7.3–3, the M-CARD SHALL keep SDO (pin 60) low. If SDO has not gone active after 5 seconds, the Host may use this as an indication that there is a problem with the M-Mode. Note that in S-CARD/Host interface, only the READY line has significance, and SDO has no meaning.

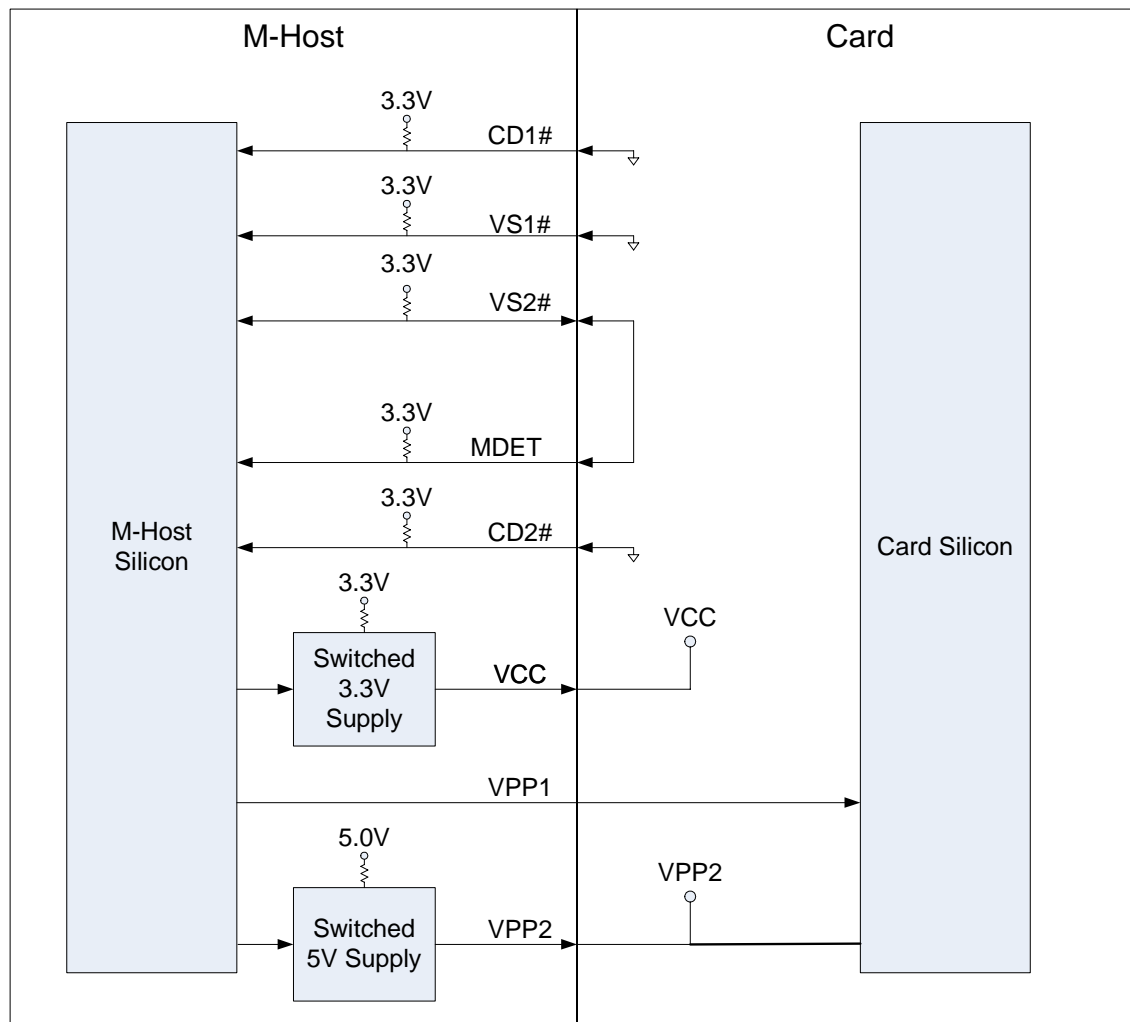
If an M-CARD is inserted into an S-Host, the Card SHALL sense the state of VPP1 and VPP2 and if they both are High, the Card SHALL begin S-Mode Start-up as defined in Section 7.2 of this document.

The following table summarizes the VPP pin configurations, and associated Card operating modes.

**Table 7.3-3 - VPP Pin Configurations, and Associated Card Operating Mode**

VPP1	VPP2	Card Configuration
Low	High	M-Mode
Low	Low	Reserved
High	Low	Reserved
High	High	S-Mode

Figure 7.3-2 shows the Card type detection and identification signals for Host capable of operating in M-Mode.

**Figure 7.3-2 - Card Type Detection Signals**

### 7.3.5 Card Information Structure

#### 7.3.5.1 S-Mode

The Card Information Structure (CIS) SHALL be readable whenever the Card is powered, the Card has been reset by the Host in accordance with section 4.12.1 of [PCMCIA2], the Card is asserting the READY signal, and the Card Personality Change has not occurred. The CIS contains the information needed by the Host to verify that a Card has been installed. After the Card Personality Change, the CIS SHALL no longer be available.

The following table is the minimum set of tuples required for the CableCARD.

**Table 7.3–4 - CIS Minimum Set of Tuples**

CISTPL_LINKTARGET
CISTPL_DEVICE_0A
CISTPL_DEVICE_0C
CISTPL_VERS_1
CISTPL_MANFID
CISTPL_CONFIG
CISTPL_CFTABLE_ENTRY
CISTPL_NO_LINK
CISTPL_END

The list of all of the S-CARD Attribute and Configuration Registers can be found in Annex D of this document.

#### **7.3.5.1.1 Operation After Invoking CableCARD Personality Change**

After the correct value is written into the configuration register, the Card SHALL wait a minimum of 10 usec before switching from the PCMCIA to Card interface.

#### **7.3.5.2 M- Mode**

In M-Mode operation, the CIS structure is not read by the Host, and does not need to be presented by the Card.

### **7.3.6 MPEG Transport Interface**

The MPEG interface consists of an input clock (MCLKI for S-Mode and MICKL for M-Mode), an input start of packet signal (MISTRT), an eight-bit input bus (MDI[7:0]), an output clock (MCLKO for S-Mode and MOCLK for M-Mode), an output start of packet signal (MOSTRT), and an eight-bit output bus (MDO[7:0]). Note that ‘input’ and ‘output’ labels are from the perspective of the Card.

For M-Mode, MDI[7:0] data and MISTRT are clocked into the Card on the rising edge of MICKL. Similarly, MDO[7:0] and MOSTRT are clocked into the Host on the rising edge of MOCLK.

MOCLK SHALL be derived from MICKL.

The MISTRT and MOSTRT signals indicate the start of an MPEG packet from the Card to the Host. They are asserted at the same time as the first byte of the CableCARD MPEG Packet header.

#### **7.3.6.1 S-Mode**

The CHI SHALL support independent both-way logical connections for the Transport Stream and for commands. The CHI SHALL accept an MPEG-2 Transport Stream, consisting of a sequence of Transport Packets, either contiguously or separated by null data. The returned Transport Stream may have some of the incoming transport packets returned in a descrambled form. The Transport Stream Interface is subject to the following restrictions:

1. When the Card is the source of a transport stream its output SHALL comply with [ISO13818-9].
2. Each output packet SHALL be contiguous if the module is the source of the packet or the input packet is contiguous.
3. The Card SHALL introduce a constant delay when processing an input transport packet, with a maximum delay variation (tmdv) applied to any byte given by the following formula:

$$tmdv_{max} = (n * TMCLKI) + (2 * TMCLKO).$$

And-

$tmdv_{max} \leq 1$  microsecond when  $n = 0$

Where:

tmdv = Module Delay Variation

n = Number of gaps present within the corresponding input transport packet

TMCLKI = Input data clock period

TMCLKO = Output data clock period

A 'gap' is defined to be one MCLKI rising edge for which the MIVAL signal is inactive.

All Hosts are strongly recommended to output contiguous transport packets, as packets arrive synchronously with the clock, but not necessarily continuously.

Inter-packet gaps may vary considerably.

4. All interfaces SHALL support a minimum byte transfer clock period of 110 ns.
5. The CHI SHALL transfer commands as defined by the appropriate Transport Layer part of this document in both directions. The data rate supported in each direction SHALL be at least 3.5 Megabits/sec.

#### **7.3.6.1.1 Transport Stream Interface – Transport Layer**

The transport layer used is the same as the MPEG-2 System transport layer. Data traveling over the transport stream interface is organized in MPEG-2 Transport Packets. The whole MPEG-2 multiplex is sent over this transport stream interface and is received back fully or partly descrambled. If the packet is not scrambled, the Card returns it as is. If it is scrambled and the packet belongs to the selected service and the Card can give access to that service, then the Card returns the corresponding descrambled packet with the transport\_scrambling\_control flag set to '00'. If scrambling is performed at Packetized Elementary Stream (PES) level, then the module reacts in the same way and under the same conditions as above, and returns the corresponding descrambled PES with the PES\_scrambling\_control flag set to '00'.

The transport packet and the PES packet are completely defined in the MPEG-2 System specification [ISO13818-1].

#### **7.3.6.1.2 Transport Stream Interface – Upper Layers**

Apart from the Packetized Elementary Stream, any layering or structure of the MPEG-2 data above the Transport Stream layer is not relevant to this specification. However the specification does assume that the Card will find and extract certain data required for its operation, such as ECM and EMM messages, directly from the Transport Stream.

#### **7.3.6.1.3 Transport Stream Interface – Link Layer**

There is no Link Layer on the Transport Stream Interface. The data is in the form of consecutive MPEG-2 Transport Packets, possibly with data gaps within and between Transport Packets.

#### **7.3.6.2 M-Mode MPEG Flow Model**

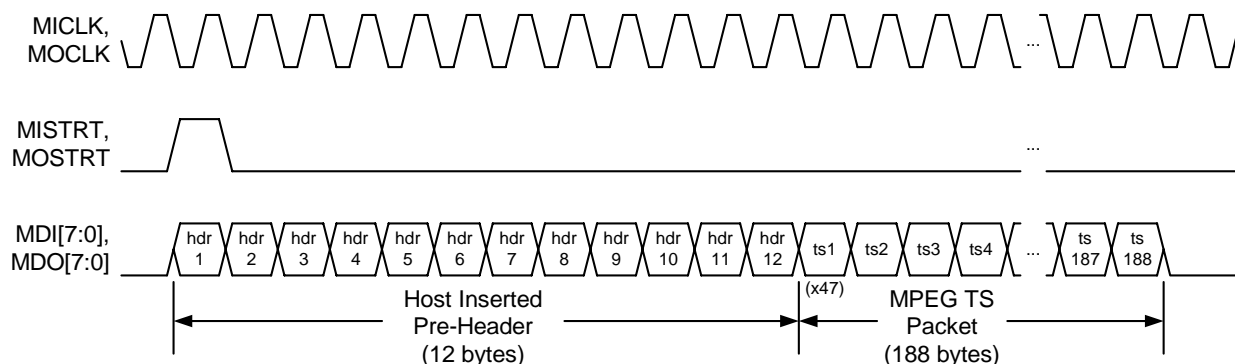
A CableCARD MPEG Packet (CMP) consists of a 188-byte MPEG packet with a 12 byte header pre-appended to MPEG packet. Each CMP packet header and packet SHALL be transmitted with no gaps after the packet start signal is active. The packet SHALL consist of 200 contiguous bytes. In other words, the packets SHALL be buffered.

Each packet header and packet SHALL have a fixed delay through the Card with no more than one MICLK period worth of jitter.

The M-CARD SHALL keep track of the number of bytes received, starting with the packet start signal, and form a packet from the first 200 bytes. During the next byte, if the packet start signal is active, that marks the beginning of the next packet, otherwise, the Card SHALL discard the sampled data until the next packet start signal is active.

If the Card receives less than 200 bytes prior to MISTRT being asserted, it is not required to pass that packet to the Host and may drop the bytes. However, this SHOULD NOT affect the delay through the Card for any other packets.

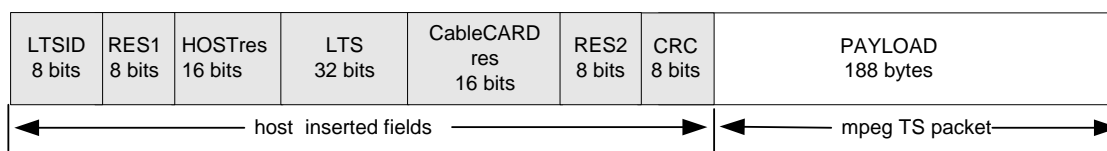
Figure 7.3-3 shows an example of a single CMP packet transmitted across the interface.



**Figure 7.3-3 - CMP Diagram**

### 7.3.6.2.1 M-Mode MPEG Transport Stream Pre-Header

A 12-byte field SHALL be pre-pended by the Host to each MPEG packet sent across the CHI. This pre-header provides identification information to allow packets from multiple transport streams to be multiplexed prior to delivery to the Card. The data format is shown in Figure 7.3-4; the transport streams are identified by local transport stream IDs (LTSID), which are inserted by the Host. This LTSID is not required to be the same as the QAM transport stream ID. These 8-bit IDs in the transport packet pre-header allow for correlation between the command APDUs and the transport streams.



**Figure 7.3-4 - M-Mode MPEG Transport Stream Pre-Header**

#### LTSID

Local Transport Stream ID – Each packet in a given transport stream SHALL be tagged with the same unique LTSID, in order to allow multiple transport streams to be multiplexed, transferred across the transport interface, correctly decrypted by the Card, and de-multiplexed, and correctly routed at their destination. The Host SHALL be responsible for generating the LTSID value. It is not required to be the same Transport Stream ID as assigned to the QAM transport stream. The Card SHALL NOT modify it.

#### Res1, Res2

Two 8-bit fields reserved for future use. The Card and Host SHOULD ignore these fields. The default value for both is 0x00.

<b>Host_reserved</b>	A 16-bit field, containing data generated by the Host, identifying additional characteristics of the transport packets. The use of this field is optional for the Host. The Card SHALL NOT modify the values in this field.
<b>LTS</b>	Local Time Stamp – A 32-bit local time stamp, whose value is set by the Host. The Card SHALL NOT modify this value. The Host MAY use the local time stamp to manage MPEG timing of the packets received from the Card. Setting a value in this field is optional for the Host.
<b>CableCARD_reserved</b>	A 16-bit field. The default value is 0x00. The usage of this field is optional for the Card.
<b>CRC</b>	Cyclic Redundancy Check – An 8-bit value calculated and inserted by the Host to provide the ability to check that the LTSID, CableCARD_reserved, Host_reserved, and LTS are transferred across the transport interface without error. If the Card inserts data into the CableCARD_reserved field, it SHALL recalculate the CRC for those packets. The CRC is calculated across the 11 bytes of the pre-header. The CRC polynomial is:

CRC-8	$x^8+x^7+x^6+x^4+x^2+1$	8
-------	-------------------------	---

**Figure 7.3-5 - CRC Polynomial**

Note: A detailed explanation of the CRC Polynomial can be found in Annex C.

## 7.4 Electrical Specifications

In order to remain compliant with the PC Card standard and to simplify the Host and Card implementations, and regardless of the powering state of the Host (i.e., active or standby), the Host and Card SHALL implement the power characteristics as define in this document.

A Common Interface module is implemented as a variant of the 16-bit PC Card Electrical Interface, Section 4 of [PCMCIA2]. The command interface uses the least significant byte of the data bus, together with the lower part of the address bus (A0-A14), and appropriate control signals. The command interface operates in I/O interface mode. The upper address lines (A15-A25), the most significant half of the data bus (D8-D15), and certain other control signals are redefined for this interface variant.

When first plugged in and before configuration, a Card conforming to this interface specification SHALL behave as a Memory-Only device with the following restrictions:

1. Signals D8-D15 shall remain in the high-impedance state.
2. 16-bit read and write modes are not available. CE2# SHALL be ignored and interpreted by the Card as always being in the 'High' state.
3. Address lines A15-A25 SHALL NOT be available for use as address lines. The maximum address space available on the Card is limited to 32768 bytes (16384 bytes of Attribute Memory as it only appears at even addresses).
4. Signals BVD1 and BVD2 SHALL remain 'High'.

### 7.4.1 DC Characteristics

#### 7.4.1.1 S-Mode

In order to remain compliant with the PC Card standard and to simplify the Host and Card implementations, and regardless of the powering state of the Host (i.e., active or standby), the following power management features are required.

- The Host SHALL permanently supply 3.3V on the VCC pins. The Host SHALL be capable of supplying up to a maximum of 1 amp total on the VCC pins (500 mA each) at 3.3 VDC per Card supported.

- The Host SHALL supply 5V on the VPP pins if requested by the Card CIS. The Host SHALL be capable of supplying up to 250 mA total on the VPP pins (125 mA each) at 5 VDC per Card supported.
- The Host SHALL continuously supply 3.3V on the VPP pins upon Host power-up and also when a Card is not installed. When a Card is installed, if the voltage sense pins are set as required per the CHI Specification, the Host SHALL supply 5 V on the VPP pins only if requested by the Card CIS. Otherwise, the Host SHALL continue to supply 3.3 V on the VPP pins while the Card is installed. Upon removal of a Card, the Host SHALL revert to or continue to supply 3.3V on the VPP pins. The Host SHALL be capable of supplying up to 250 mA total on the VPP pins (125 mA each) at 5 VDC per Card supported.
- If the Host receives a value of 0x3 in the Power field of the Feature Selection Byte (TPCE\_FS) from the Card according to section 6.1.2 of [SCTE28], the Host SHALL NOT be required to support the separate nominal voltage parameter descriptors in the power descriptor structures for VPP1 and/or VPP2. If the Host does not support the Power Field value of 0x3h, then it SHALL continue to supply a nominal voltage of +3.3V to both the VPP1 and VPP2 pins.
- The Card SHALL only support the value of 0x2 in the Power field of the Feature Selection Byte (TPCE\_FS) and the associated parameter descriptor according to section 3.3.2.3 of [PCMCIA4] if the Card requires a switched nominal voltage level of +5V on the VPP lines.
- There is no standby power mode for the Card.
- The Card SHALL draw no more than 2.5 watts averaged over a period of 10 seconds.
- The Host OOB Receive circuitry SHALL continue to operate in all powering states of the Host.
- The Host SHALL support hot insertion and removal of the Card.
- The Card SHALL implement the mechanical Low Voltage Keying.
- The Card SHALL force VS1# (pin 43) to ground and VS2 (pin 57) to high impedance until it switches to the CableCARD device Custom Interface mode.
- The Card SHALL support 3.3V hot insertion.
- The Card does not have to meet the requirement of section 4.12.2 of [PCMCIA2] to limit its average current to 70 mA prior to the CableCARD device Personality Change (writing to the Configuration Option Register).

#### **7.4.1.2 M-Mode**

- The Host SHALL provide 3.3 V on VCC and 5V on VPP2 with the DC Characteristics as defined in Table 7.4–1. The Host SHALL apply 3.3 V on VCC and 5V on VPP2 as defined in Section 7.3.4.2.
- The Host SHALL be capable of providing up to 1 Amp total on the VCC pins (500 mA each) per Card supported.
- The Host SHALL be capable of providing up to 125 mA on the VPP2 pin.
- There is no standby power mode for the Card.
- The Card SHALL NOT draw more than 2.5 watts averaged over a period of 10 seconds.
- Pin type LogicPC - DC Signal levels as defined in Table 4-15 DC Specifications for 3.3V Signaling in Section 4.7.1 of [PCMCIA2] Release 8.0.
- The Host OOB circuitry SHALL continue to operate in all powering states of the Host.
- Upon removal of the Card, when operating in M-Mode, the Host SHALL detect the state of the CD1# and CD2# pins and power-down the VCC and VPP interface pins, and upon Card insertion, determine the Card type prior to applying VCC and VPP power.

**Table 7.4–1 - M-Mode Power Supply DC Characteristics**

Symbol	Parameter	Min	Max	Units	Notes
VCC	Supply Voltage	3.0	3.6	V	1
I <sub>CC</sub>	Supply Current	-	1.0	A	2
VPP2	Supply Voltage	4.75	5.25	V	1
I <sub>PP2</sub>	Supply Current	0	0.125	A	2
Notes: 1. There is no standby power mode for the Card. 2. The Host SHALL be capable of providing up to 1 Amp total on the VCC pins (500 mA each) per Card.					

### 7.4.1.3 Signaling Interfaces

#### 7.4.1.3.1 S- Mode

The Card and Host SHALL support the Signal Interface as defined in section 4.9 of [PCMCIA2]. Support by Hosts for overlapping I/O address windows as defined in section 4.9.3.2 of [PCMCIA2] is optional. Cards SHALL use an independent I/O address window 4 bytes in size.

The DC characteristics of the interface signals in S-Mode are defined in Section 7.3.3.4 of this document with the exception that when the M-CARD is operating in S-Mode MICK (A12 pin 21), will have the characteristics of CCLK signal type.

The Host SHALL detect Card insertion and removal using CD1# and CD2# and the Card SHALL provide a means of allowing the Host to detect Card insertion and removal as described in Section 4.10 of [PCMCIA2].

Cards SHALL NOT implement or require the battery voltage detect function and support for it by Host is optional.

The Card and the Host SHALL implement the I/O function as defined in Section 4.13 of [PCMCIA2] except that the Card SHALL only use 8-bit read and write modes.

The Card SHALL implement the Card Configuration as identified in 4.15 and 4.15.1 of [PCMCIA2].

#### 7.4.1.3.2 S-Mode and M- Mode Signal Types

Table 7.4–2 shows the signal type for each of the interface signals when used in S-Mode and M-Mode.



**Table 7.4–2 - Card Signal Types by Mode**

PC Card Memory-Only Signals	S-Mode Signals	M-Mode Signals	Signal Type
A13, A12	Unused	MOCLK, MICKL	CCLK
A[25:17]	MDI[7:0], MISTRT	MDI[7:0], MISTRT	LogicCB
D[15:8], BVD1	MDO[7:0], MOSTRT	MDO[7:0], MOSTRT	LogicCB
A[9:4]	DRX, CRX, CTX, ITX, QTX, ETX	DRX, CRX, CTX, ITX, QTX, ETX	LogicPC
A[2:0], RFU	Unused, A[1:0], INPACK#	SCTL, SCLK, SDI, SDO,	LogicPC
VS1#, VS2#	VS1#, VS2#	VS1#, VS2#	Sense <sup>1</sup>
RESET	RESET	RESET	LogicPC
WP	IOIS16#	MDET	LogicPC <sup>1</sup>
VPP1, VPP2	VPP1, VPP2	VPP1, VPP2	VPP1, VPP2
CD[2:1]#	CD[2:1]#	CD[2:1]#	Sense
Note: 1. VS2# is also tied directly to MDET, thus MDET does not need to be sourced or driven from the Card but sourced or driven by the Host via VS2#. 2. The LogicPC signal type applies to VPP1 and VPP2 only during Card type identification. Otherwise, VPP1 and VPP2 have the supply characteristics described in Section 7.4.			

Table 7.4–3 shows the DC characteristics of the signal interfaces for the Card.

**Table 7.4–3 - DC Signal Requirements**

Signal Type	DC Signal Requirements
CCLK	DC Signal levels as defined in Table 5-7 DC Specifications for 3.3V Signaling in Section 5.3.2.1.1 of [PCMCIA2] Release 8.0.
LogicCB	DC Signal levels as defined in Table 5-7 DC Specifications for 3.3V Signaling in Section 5.3.2.1.1 of [PCMCIA2] Release 8.0.
LogicPC	DC Signal levels as defined in Table 4-15 DC Specifications for 3.3V Signaling in Section 4.7.1 of [PCMCIA2] Release 8.0.
Sense	Sense signals as defined in [PCMCIA2].

Table 7.4–4 is the DC signaling characteristics for the “LogicPC” signaling level.

**Table 7.4–4 - DC Signaling Characteristics for the “LogicPC” Signaling Level**

Symbol	Parameter	Min	Max	Units	Notes
VCC	Supply Voltage	3.0	3.6	V	
V <sub>IH</sub>		2.0	VCC + 0.3	V	
V <sub>IL</sub>		-0.3	0.8	V	
V <sub>OH</sub>		2.4 (VCC-0.2)		V	1
V <sub>OL</sub>			0.4 (0.2)	V	1
Note: All logic levels per JEDEC 8-1B. This table is for reference only. 1. For CMOS Loads					

Table 7.4–5 is the DC signaling characteristics for the “LogicCB” and “CCLK” signaling level.

**Table 7.4–5 - DC Signaling Characteristics for the “LogicCB” Signaling Level**

Symbol	Parameter	Condition	Min	Max	Units	Notes
VCC	Supply Voltage		3.0	3.6	V	
V <sub>IH</sub>			0.475V <sub>CC</sub>	VCC + 0.5	V	
V <sub>IL</sub>			-0.5	0.325VCC	V	
I <sub>IL</sub>		0 < V <sub>in</sub> < VCC		+/-10	uA	
V <sub>OH</sub>		I <sub>out</sub> = -150uA	0.9VCC		V	
V <sub>OL</sub>		I <sub>out</sub> = 700uA		0.1VCC	V	
Card	Card Input Pin Capacitance		5	17	pF	
Chost	System Load Capacitance		5	22	pF	
Note: All logic levels per [PCMCIA2]. This table is for reference only.						

Table 7.4–6 indicates the requirements for Pullups and Pulldowns for S-Mode and M-Mode Operation.

**Table 7.4–6 - CableCARD and Host Pullups and Pulldowns**

Item	Signal	S-CARD	M-CARD	S-Host	M-Host	Notes
Card Detect	CD[2:1]#			pullup to Host 3.3V R. $\geq$ 10K	pullup to Host 3.3V R. $\geq$ 10K	6
Voltage Sense	VS[2:1]#			pullup to Host 3.3V 10K $\leq$ R. $\leq$ 100K	pullup to Host 3.3V 10K $\leq$ R. $\leq$ 100K	
Card Type Detect	MDET				pullup to Host 3.3V R $\geq$ 100K	
Control Signal	RESET	pullup to VCC R $\geq$ 100K	pullup to VCC R $\geq$ 100K			3
MPEG Interface	MICLK, MISTR, MDI[7:0]		pulldown R $\geq$ 100K			1
	MOCLK		pulldown R $\geq$ 100K			1
	MOSTRT				Pullup to Host 3.3V R. $\geq$ 10K	1
	MDO[7:0]	pulldown R $\geq$ 100K	pulldown R $\geq$ 100K			1
OOB Interface	CRX, DRX, CTX	pulldown R $\geq$ 100K	pulldown R $\geq$ 100K			1, 5
	ITX, QTX, ETX	pulldown R $\geq$ 100K	pulldown R $\geq$ 100K			1, 4
CPU Interface	SCTL, SCLK, SDI		pulldown R $\geq$ 100K			1, 5
	SDO			pullup to Host 3.3V R. $\geq$ 10K		2, 4

**Notes:**

1. Due to PC Card Requirement for pullups and pulldowns on the Memory Interface.
2. S-Host has R  $\geq$  10K pullup required for INPACK#. The M-Host is responsible for providing signal conditioning (i.e., a pullup) in a manner that meets the DC and AC requirements of this signal.
3. Note 3 of Table 4-16 of in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
4. Note 4 of Table 4-16 of in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
5. Note 5 of Table 4-16 of in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
6. The CableCARD device SHALL force VS1# (pin 43) to ground and VS2 (pin 57) to high impedance until it switches to the CableCARD device Custom Interface mode.

## 7.4.2 AC Characteristics

### 7.4.2.1 S-Mode

All Card signal requirements and timing requirements SHALL comply with Table 7.4–7 and Figure 7.4-1 and Figure 7.4-2, and SHALL be measured with no less than the maximum load of the receiver as defined in table 4-16 of [PCMCIA2].

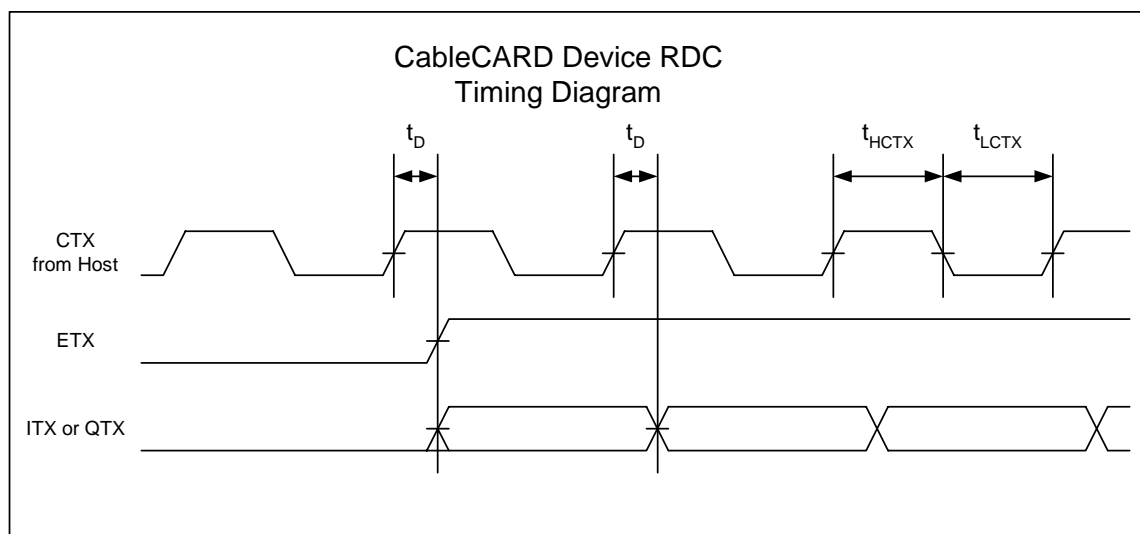
The PC Card A7, A6, and A5 pin definitions have been modified to QTX, ETX, and ITX. These pins will be driven by the Card and will have Data Signal characteristics per table 4-16 of [PCMCIA2]. Additionally, the signals MOVAL and MOSTRT will be driven by the Card and will have Data Signal characteristics per table 4-16 of [PCMCIA2]. The remaining signals follow the signal type assignments as listed in table 4-16 of [PCMCIA2].

All signal voltage levels are compatible with normal 3.3V CMOS levels.

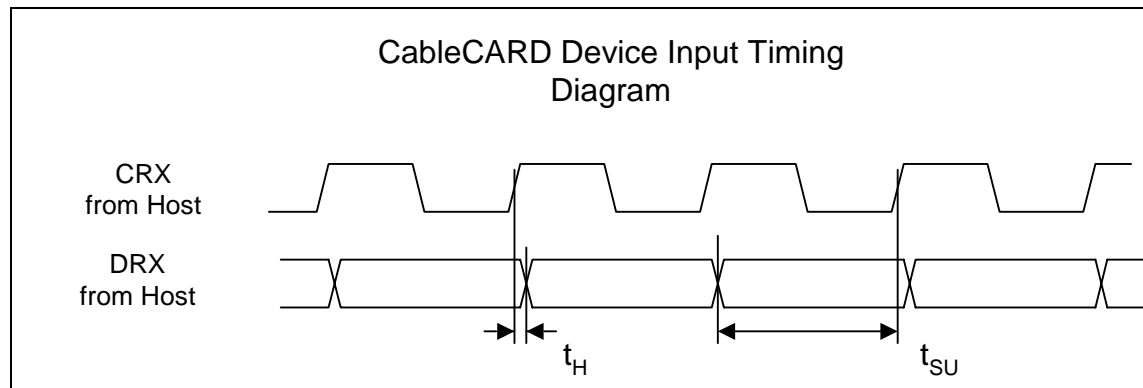
**Table 7.4-7 - S-Mode Signal Parameters**

Parameter	Signal	Unit	Min	Typ	Max	Conditions
Frequency	CTX	kHz			3088	
Frequency	CRX	kHz			3088	
Clock High Time ( $T_{HIGH}$ )	CTX, CRX	ns	250			Notes 1, 2
Clock Low Time ( $T_{LOW}$ )	CTX, CRX	ns	250			Notes 1, 2
Delay ( $t_D$ )	ETX, ITX, QTX	ns	5		180	Notes 1, 2
Set-up ( $T_{su}$ )	DRX	ns	10			From time signal reaches 90% of high level (rising) or 10% of high level (falling) until CRX mid-point transition
Hold ( $T_h$ )	DRX	ns	5			From CRX mid-point transition until signal reaches 10% of high level (rising) or 90% of high level (falling)
Notes: 1. Refer to Figure 7.4-1 - CableCARD Device Output Timing Diagram. 2. AC Timing is measured with Input/Output Timing Reference level at 1.5V.						

The AC Timing characteristics of the OOB FDC and RDC timing for the Card operating in S-Mode and M-Mode is illustrated in Figure 7.4-1 and Figure 7.4-2.



**Figure 7.4-1 - CableCARD Device Output Timing Diagram**



**Figure 7.4-2 - CableCARD Device Input Timing Diagram**

#### **7.4.2.2 M-Mode**

When the Card is operating in M-Mode, the AC signaling characteristics of CCLK, LogicPC and Logic CB are described below.

- Pin type CCLK has AC Signal levels as defined in Table 5-9, AC Specifications for 3.3V Signaling (CCLK), in Section 5.3.2.1.4 of [PCMCIA2] Release 8.0.
- Pin type LogicCB- AC Signal levels as defined in Table 5-8, AC Specifications for 3.3V Signaling, in Section 5.3.2.1.2 of [PCMCIA2] Release 8.0.

#### 7.4.2.2.1 Power and Reset Timing

The power-on sequence timing is similar to the 16-bit PC card standard, with the exception that the MCLKI and the SCLK are running prior to RESET being released and VPP2 will be 5V instead of 3.3V. Figure 7.4-3 and Table 7.4-8 show the power up timing requirements.

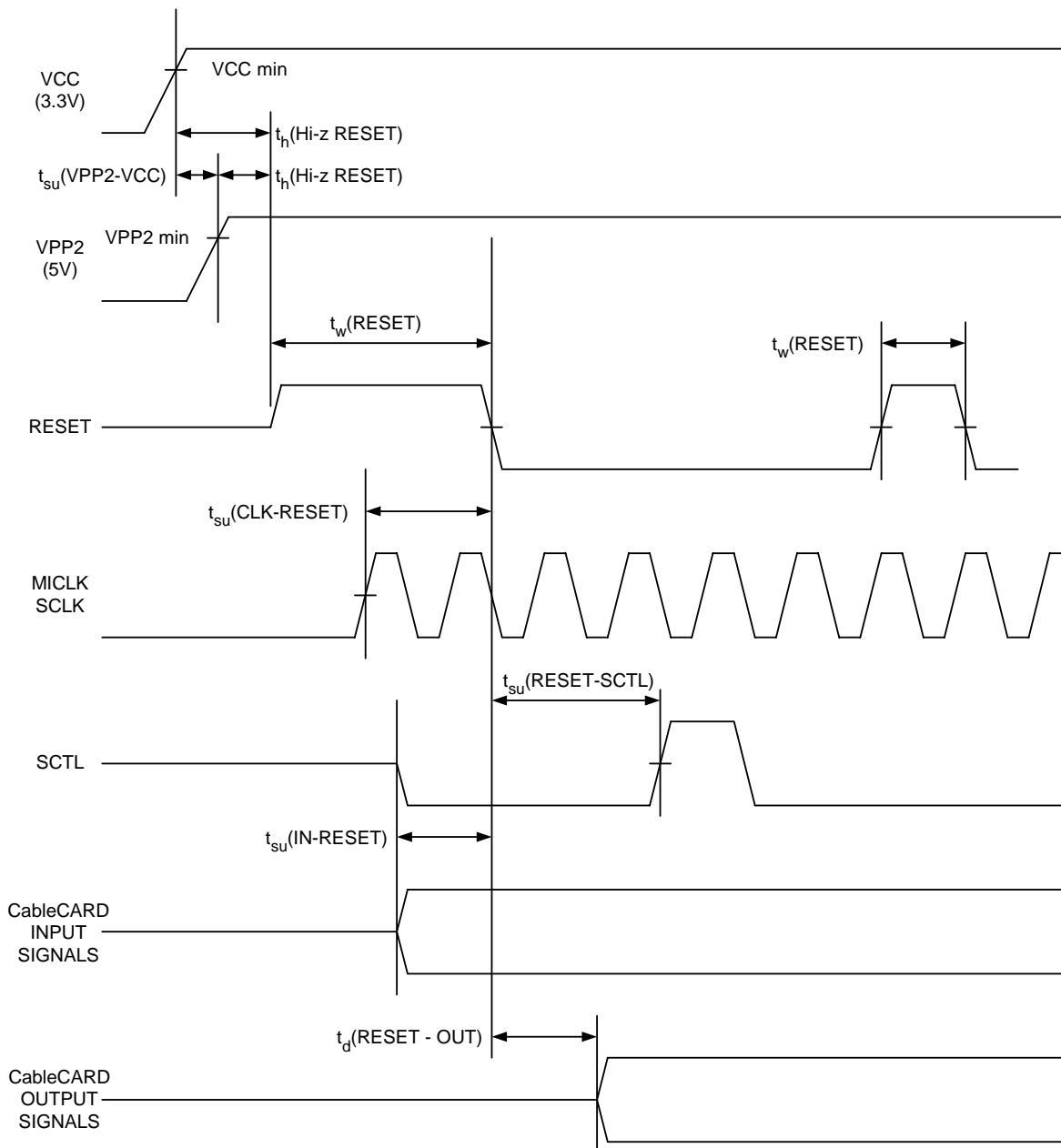


Figure 7.4-3 - M-CARD Power-On and Reset Timing Diagram

**Table 7.4–8 - M-CARD Power-On and Reset Timing Requirements**

Symbol	Parameter	Min	Max	Units	Notes
$t_{su}(VPP2-VCC)$	VCC valid to VPP2 valid	50		us	1
$t_h(Hi-z\ RESET)$	VCC and VPP2 valid to RESET assert	1		ms	
$t_w(RESET)$	Reset pulse width	10		us	
$t_{su}(CLK-RESET)$	Clock valid to RESET negate	0		ms	
$t_{su}(RESET-SCTL)$	Reset Negate to CPU Interface Active	20		ms	2
$t_{su}(IN-RESET)$	Reset Negate to CPU Interface Active	0		ms	2
$t_d(RESET-OUT)$	Reset Negate to M-CARD Output Signals Valid	0	20	ms	3
Notes: 1. VPP2 SHALL NOT exceed VCC=0.3V until VCC has reached VCC min as defined in Table 7.4–1. 2. The M-CARD input logic signals are Hi-z from when VCC and VPP2 are applied until RESET is asserted. 3. The M-CARD output logic signals are Hi-Z from when VCC and VPP2 are applied until RESET is de-asserted/negated.					

**7.4.2.2.2 MPEG Packet Jitter**

The Host SHALL be responsible for multiplexing and demultiplexing the transport packets, in a manner to minimize jitter of the MPEG PCR. However, packets sent across the interface from the Host to the Card SHOULD be returned to the Host with a uniform, fixed delay from receipt to transmission back to the Host. The fixed delay MAY vary by up to, but not exceed, one MICK period (MICK is the MPEG transport interface input clock i.e.,  $\pm 1$  MICK period).

All transport stream packets sent across the interface will be returned in the same order in which they are received by the Card.

**7.4.2.2.3 MPEG Transport Timing**

The setup and hold times of the MPEG transport interfaces will be similar to the Cardbus data to CCLK Timing Parameters.

**Table 7.4–9 - M-CARD MPEG Transport Timing**

Symbol	Parameter	Min	Max	Units	Notes
$t_{cyc}$	MICK and MOCLK Cycle Time	37.00	37.074	ns	1, 2, 3
$t_{high}$	MICK and MOCLK High Time	15		ns	3
$t_{low}$	MICK and MOCLK Low Time	15		ns	3
$t_{su}$	Input Setup time of MDI[7:0] and MISTRT to MICK and Input Setup Time of MDO[7:0] and MOSTRT to MOCLK	7		ns	4
$t_h$	Input Hold time of MICK to MDI[7:0] and MISTRT and Input Hold Time of MOCLK to MDO[7:0] and MOSTRT	0		ns	4

Symbol	Parameter	Min	Max	Units	Notes
$t_{val}$	MICLK to and MOCLK to Output Signal Valid Delay.	2	18	ns	4
Notes: 1. MOCLK will be derived directly from MICKL. 2. The Nominal Frequency is 27MHz. As specified, the Card operating in M-Mode will be required to operate at 27MHz +/- 1000ppm although the M-Host may be required to operate at tighter tolerances to maintain MPEG timing. 3. See Figure 5-28, CardBus PC Card Clock Waveform, of [PCMCIA2] Release 8 for reference levels. 4. See Figure 5-30, Input Timing Measurement Conditions, and Table 5-12, Measurement and Test Condition Parameters, of [PCMCIA2] Release 8 for reference levels.					

#### 7.4.2.2.4 S-Mode CPU Interface Timing

The CPU functions for the Card operating in S-Mode are defined in Section 7.6.1.

#### 7.4.2.2.5 M-Mode CPU Interface Timing

- **SCLK** - The clock SHALL operate at a nominal rate of 6.75 MHz (27 MHz/4).
- **SCTL** - The serial control signal SHALL change on the falling edge of the SCLK signal.
- **SDI** - The serial Host data SHALL change on the falling edge of the SCLK signal. The Card SHALL input the data on the rising edge of the SCLK signal.
- **SDO** - The serial Card data SHALL change on the falling edge of the SCLK signal. The Host SHALL input the data on the rising edge of the SCLK signal.

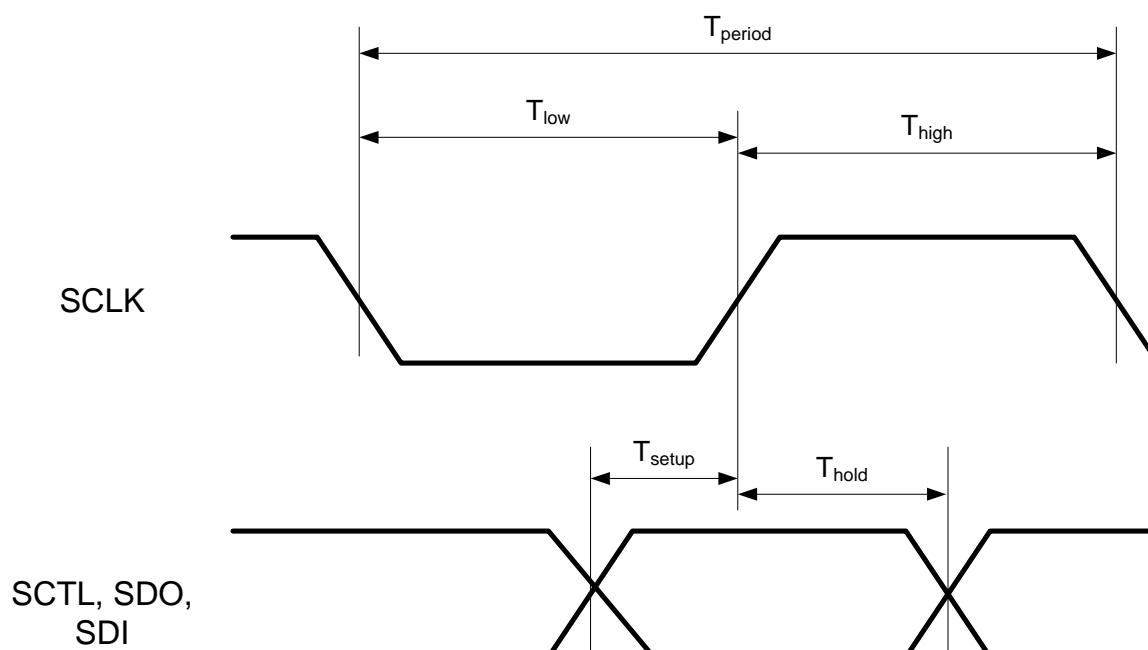


Figure 7.4-4 - M-Mode Serial Interface Timing Diagram



**Table 7.4–10 - M-Mode Serial Interface Timing**

Signal	Value	Nominal	Max	Min	Unit
SCLK	Frequency	6.75	7.00	6.50	MHz
	T <sub>high</sub>	74	88	59	ns
	T <sub>low</sub>	74	88	59	ns
SCTL	T <sub>setup</sub>	n/a	n/a	7	ns
	T <sub>hold</sub>	n/a	n/a	0	ns
SDI	T <sub>setup</sub>	n/a	n/a	7	ns
	T <sub>hold</sub>	n/a	n/a	0	ns
SDO	T <sub>setup</sub>	n/a	n/a	7	ns
	T <sub>hold</sub>	n/a	n/a	0	ns
Notes: Minimum times are specified with 0 pF equivalent load; maximum times are specified with 30 pF equivalent load. Actual test capacitance MAY vary but results SHOULD be correlated to these specifications. The duty cycle of SCLK SHALL be no less than 40%, no more than 60%.					

## 7.5 Mechanical Specifications

### 7.5.1 Form Factor

The mechanical design of the Card SHALL follow either the PC Card or CardBus specifications called out in [PCMCIA3]. Additionally, any future modifications to the physical specification, which are backwards compatible, may be implemented.

The Card SHALL comply with the Type II PC Card Package Dimensions with Low Voltage Keying as shown in Figure 11-3 of [PCMCIA3].

The M-Card SHALL include the CardBus/CardBay PC Card Recommended Connector Grounding as shown in Figure 11-40 [PCMCIA3]. Visual identification to distinguish between a Single-Stream CableCARD device and a Multi-Stream CableCARD device will be via the label on the Card.

### 7.5.2 Connector

The Card connector and guidance SHALL comply with the PCMCIA Cardbus Type II connector and guidance defined in [PCMCIA3].

### 7.5.3 Environmental

When the Host and Card are operating at room temperature and humidity, no greater than 25°C ambient temperature, no greater than 95% RH non condensing, with a reference power load Card, the Host and Card SHALL NOT allow any external protruding surface point hotter than 50°C for metallic, and 60°C for non-metallic surfaces, and no non-accessible surface point hotter than 65°C.

### 7.5.4 PC Card Guidance

The Host SHALL have PC Card guidance described in section 5 of [PCMCIA3].

### 7.5.5 Grounding/EMI Clips

The Card SHALL have grounding and EMI clips described in section 6 of [PCMCIA3].

### 7.5.6 Connector Reliability

The Host SHALL have connector reliability described in section 7 of [PCMCIA3].

### 7.5.7 Connector Durability

The Host SHALL have connector durability described in section 8.2 (harsh environment) of [PCMCIA3].

### 7.5.8 PC Card Environmental

The Card SHALL meet or exceed all environmental tests of Environmental Resistance Section described in section 9 of [PCMCIA3]. With regard to temperature specification the Card SHALL operate at up to 55°C, as defined in the PC Card Specification. This is primarily to enable reliable battery operation in Cards. In order to facilitate this operating environment limit the Host SHALL limit the temperature rise between the ambient environment outside the Host and the ambient environment surrounding the Card to 15°C when the Card is dissipating its full rated power. Note that this may not guarantee that the 55°C limit for Card is met in all environmental conditions otherwise acceptable to the Host. Card designers SHALL minimize the risk of misoperation of Cards containing batteries by following the recommendations in the PC Card standard on battery placement and Host designers should be aware of these recommendations when doing Host thermal design. Note that designers will also have to conform to any relevant mandatory safety specifications with regard to Card temperature.

## 7.6 CPU Interface

### 7.6.1 S-Mode

With OOB traffic included, the Card requires more bandwidth and connections on the CPU Interface than are supported by the Data Channel alone. Two communication paths SHALL share the same pins on the PC Card connector.

**Data Channel** – This channel is compliant as defined below, plus the interrupt mode extension. Card applications will use this path when they require support from Host resources.

The hardware interface consists of four registers occupying 4 bytes in the address space on the PC Card interface. Byte offset 0 is the Data Register. This is read to transfer data from the Card and written to transfer data to the Card. At byte offset 1 are the Control Register and Status Register. Reading at offset 1 reads the Status Register, and writing at offset 1 writes to the Control Register. The Size Register is a 16-bit register at byte offsets 2 and 3. Offset 2 is the Least Significant half and offset 3 the Most Significant half. The register map is shown in Figure 7.6-1.

Only two address lines, A0 and A1, are decoded by the interface. The Host designer is free to place this block of 4 bytes anywhere within its own address space by suitable decoding or mapping of other address lines within the Host.

Offset	Register
0	Data Register
1	Control/ Status Register
2	Size Register (LS)
3	Size Register (MS)

**Figure 7.6-1 - Hardware Interface Registers**

The Status Register looks like Figure 7.6-2.

bit	7	6	5	4	3	2	1	0
	DA	FR	R	R	R	R	WE	RE

**Figure 7.6-2 - Status Register**

DA (Data Available) is set to '1' when the Card has some data to send to the Host.

FR (Free) is set to '1' when the Card is free to accept data from the Host, and at the conclusion of a Reset cycle initiated by either a Card hardware reset, or by the RS command.

R indicates reserved bits. They read as zero.

WE (Write Error) and RE (Read Error) are used to indicate length errors in read or write operations.

The Command Register looks like Figure 7.6-3.

bit	7	6	5	4	3	2	1	0
	R	R	R	R	RS	SR	SW	HC

**Figure 7.6-3 - Command Register**

RS (Reset) is set to '1' to reset the interface. It does not reset the whole Card.

SR (Size Read) is set to '1' to ask the Card to provide its maximum buffer size. It is reset to '0' by the Host after the data transfer.

SW (Size Write) is set to '1' to tell the Card what buffer size to use. It is reset to '0' by the Host after the data transfer.

HC (Host Control) is set to '1' by the Host before starting a data write sequence. It is reset to '0' by the Host after the data transfer.

R indicates reserved bits. They SHALL always be written as zero.

For Host to Card transfers, the Host sets the HC bit and then tests the FR bit. If FR is '0' then the interface is busy and the Host must reset HC and wait a period before repeating the test. If FR is '1' then the Host writes the number of bytes it wishes to send to the Card into the Size register and then writes that number of data bytes to the Data register. This multiple write SHALL NOT be interrupted by any other operations on the interface except for reads of the Status Register. When the first byte is written the module sets WE to '1' and sets FR to '0'. During the transfer the WE bit remains at '1' until the last byte is written, at which point it is set to '0'. If any further bytes are written then the WE bit is set to '1'. At the end of the transfer the Host shall reset the HC bit by writing '0' to it.

The Host must test the DA bit before initiating the Host-to-Card cycle above in order to avoid deadlock, as the CHI does not support a single buffer implementation in the Card.

Both the Card and the Host SHALL provide a double buffer implementation.

This C code fragment illustrates the Host side process:

```
if (Status_Reg & 0x80) /* go to module-to-host transfer (see below) */
Command_Reg = 0x01;
if (Status_Reg & 0x40) {
    Size_Reg[0] = bsize & 0xFF;
    Size_Reg[1] = bsize >> 8;
    for (i=0; i<bsize; i++)
        Data_Reg = write_buf[i];
}
Command_Reg = 0x00;
```

For the Card to Host Transfers Periodically the Host tests the DA bit in the Status Register. If DA is '1' then the Host reads the Size Register to find out how much data is to be transferred. It then reads that number of data bytes from the Data register. This multiple read SHALL NOT be interrupted by any other operations on the interface except for reads of the Status Register. When the first byte is read the Card sets RE to '1' and sets DA to '0'. During the transfer the RE bit remains at '1' until the last byte is read, at which point it is set to '0'. If any further bytes are read then the RE bit is set to '1'. This C code illustrates the host side process:

```
if (Status_Reg & 0x80) {
```

```

    bsize = Size_Reg[0] | Size_Reg[1] << 8;
    for (i=0; i<bsize; i++)
        read_buf[i] = Data_Reg;
}

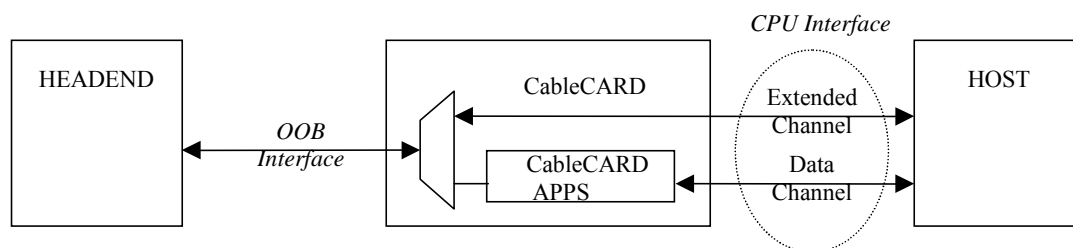
```

The bytes of the Size Register can be read or written in either order.

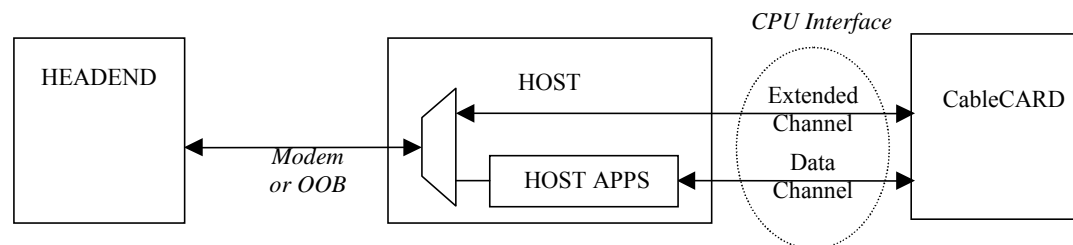
Note that this interface does not support interrupts of the Host by the Card. The Host is expected to test the DA and FR bits in the status register periodically to determine if communication is required.

**Extended Channel** – This second communication only includes physical and link layers. The purpose of the Extended Channel is to provide a communication path between the Card and the Host such that applications in one, e.g., Host, Card, can communicate with the headend via a link layer or modem function in the other Card, Host respectively. Whereas the content and format of the messages for the Data Channel are well defined, the content and format of the messages for the Extended Channel are application specific.

Depending on whether the Card or the Host is acting as the modem (or link device), the Extended Channel has a reversible function as described in Figure 7.6-4 and Figure 7.6-5.



**Figure 7.6-4 - Modem in-the-Card System Overview**



**Figure 7.6-5 - Modem in-the-Host System View**

When the Data Channel is physically activated by CE1# (Card Enable 1), the Extended Channel is enabled by CE2# (Card Enable 2).

The Extended Channel includes the same type of registers for the Command Interface. The Card enables access to the Extended Channel after the initialization phase. At this time, the CE2# signal interpretation begins, and the Extended hardware interface registers can be read and written. The signals mentioned in the table below are all inputs for the Card. The registers depicted in Figure 7.6-6 are part of the Card.

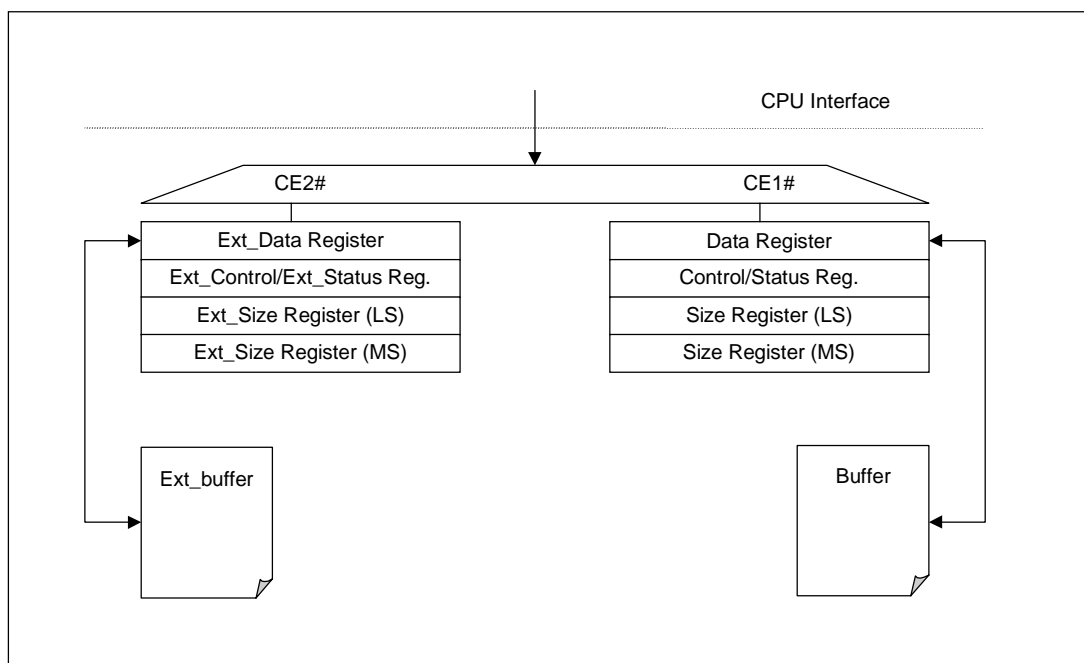
**Table 7.6-1 - Extended Interface Registers**

Extended Interface Reg.	REG#	CE2#	CE1#	A1	A0	IORD#	IOWR#
Standby mode	X	H	H	X	X	X	X
Ext_Data Write	L	L	H	L	L	H	L
Ext_Data Read	L	L	H	L	L	L	H
Ext_Command Write	L	L	H	L	H	H	L

Extended Interface Reg.	REG#	CE2#	CE1#	A1	A0	IORD#	IOWR#
Ext_Status_Reg. Read	L	L	H	L	H	L	H
Ext_Size (LS) Write	L	L	H	H	L	H	L
Ext_Size (LS) Read	L	L	H	H	L	L	H
Ext_Size (MS) Write	L	L	H	H	H	H	L
Ext_Size (MS) Read	L	L	H	H	H	L	H

The Extended Channel has its own data buffer that may have a different size than the Data Channel buffer.

Since there are two physical channels (data channel and extended channel), the behavior of the interface is defined in such a way that when the Host sets the RS\_flag on either channel, the interface is reset for both channels. Therefore, if the Host sets an RS\_flag after detection of an error condition, it should set the RS\_flag for both channels.



**Figure 7.6-6 - Map of Hardware Interface Registers**

### 7.6.1.1 Control Register Modification

The following extension to the EIA-679-B Part B Command Interface SHALL be used in order to facilitate the interrupt mode over the Data Channel and the Extended Channel.

The DA & FR bits of the Status Register should be gated onto the IREQ# line by two new Interrupt Enable bits for the Control Register: DAIE (bit 7) and FRIE (bit 6) respectively. The Control register now becomes:

**Table 7.6-2 - Control Register Definitions**

Bit	7	6	5	4	3	2	1	0
	DAIE	FRIE	R	R	RS	SR	SW	HC

RS, SR, SW and HC retain their function, as described in the [NRSSB] specification.

When set, DAIE allows any assertion of the DA (Data Available) bit in the Status register also to assert IREQ#.

When set, FRIE allows any assertion of the FR (Free) bit in the Status register also to assert IREQ#.

When IREQ# is asserted, the Host SHALL first check the data channel, and then the extended channel to determine the source of the interrupt.

### 7.6.1.2 Status Register Modification

The following extension to [NRSSB] Status Interface SHALL be used in order to allow the Card to request the initialization process to occur. A new status bit called the Initialize Interface Request (IIR) is added to bit 4 of the Status Register to allow the Card to request that the interface be re-initialized. This bit exists in both the data channel and extended channel. When the Card sets the IIR flag, the Card SHALL also reset the IIR flag when the RS flag is set.

**Table 7.6–3 - Status Register Definitions**

Bit	7	6	5	4	3	2	1	0
	DA	FR	R	IIR	R	R	WE	RE

## 7.6.2 M-Mode

When the Card is operating in M-Mode, the CPU interface consists of two logical channels, the data (or command) channel and the extended channel. The command channel is typically used for command and control transactions, while the extended channel is typically used for data transfers (SI, EAS, IP, etc.).

### 7.6.2.1 Physical Interface

The physical interface for the CPU interface is a modified SPI (Serial Peripheral Interface). Since the only connection is between the Host and the Card, the phase is fixed with the data changing on the falling edge of the clock (SCLK) and clocked in on the rising edge. A control signal (SCTL) SHALL be utilized to signal the start of a byte of data as well as the start of a new packet. Two separate data signals SHALL be used: M-Host to Card data (SDI), and Card to M-Host (SDO).

### 7.6.2.2 Packet Format

When the start of a packet occurs, the first byte is defined to be the interface query byte, which includes the interface flags defined below.

After the interface query byte, the packet count consists of two bytes, which contain the number of data bytes following in the packet. In other words, the ‘length’ does not include the first three bytes of the packet. The MSB of the packet count is transmitted first. The maximum number of data bytes in a packet is 4,096. Therefore, the three most significant bits of the 16-bit length should always be zero.

**Table 7.6–4 - CPU Interface Packet Format**

	Bit	7	6	5	4	3	2	1	0
Host	Query	X	HR	EC	L	F	DA	CR	X
	Length MSB	b7	b6	b5	b4	b3	b2	b1	b0
	Length LSB	b7	b6	b5	b4	b3	b2	b1	b0
	Data Byte(s)	b7	b6	b5	b4	b3	b2	b1	b0
M-CARD	Query	X	CR	EC	L	F	DA	ER	X
	Length MSB	b7	b6	b5	b4	b3	b2	b1	b0
	Length LSB	b7	b6	b5	b4	b3	b2	b1	b0

	Bit	7	6	5	4	3	2	1	0
	Data Bytes(s)	b7	b6	b5	b4	b3	b2	b1	b0

After the packet count, if the DA bit was set in the interface query byte, either the command channel or extended channel data SHALL follow.

### 7.6.2.3 Interface Flags

- HR**      **Host Ready:** The Host SHALL set this flag when it is ready to transmit or receive data. The Card SHALL monitor this flag and not set the CR flag until this flag is set. This flag allows the Host to control the throughput of packets from the Card.
- CR**      **CableCARD Ready:** The Card SHALL set this flag when it is ready to transmit or receive data. After the RESET signal goes inactive, this signal will indicate to the Host when the Card is ready. The Card SHALL set this flag less than 5 seconds after RESET goes inactive. This flag also allows the M-CARD to control the throughput of packets from the Host.
- EC**      **Extended Channel:** 0 = Command Channel, 1 = Extended Channel. The Host or Card SHALL use this flag to determine whether the data transmitted from the source is intended for the Command or Extended channels. It SHOULD be noted that it is possible for the Host to transmit Command Channel data while the Card is transmitting Extended Channel data or vice versa.
- L**        **Last:** Indicates to the Host/Card that the packet is the last one. Transactions are segmented into multiple packets only when the data size is greater than 4,096 bytes. Transactions that contain less than 4,096 data bytes SHALL set this flag.
- F**        **First:** Indicates to the Hosts/Card that the packet is the first one. Transactions are segmented into multiple packets only when the data size is greater than 4,096 bytes. Transactions that contain less than 4,096 data bytes SHALL set this flag.
- DA**      **Data Available:** Indicates to the Host/Card that data is available. When the Host detects that the Card has set this flag, it SHALL read the following length bytes from the Card and SHALL clock all of the remaining data in prior to indicating a start of message. The Card MAY choose to ignore the Host's DA flag.
- ER**      **Error detected:** The Host or Card has detected an error in the CPU interface. If the Host detects that the Card has set this flag, it SHALL reset the Card. The Card MAY choose to ignore the Host's ER flag.

### 7.6.2.4 Interface Model

The Host is the master of this interface. The Host SHALL always transmit the interface query byte (IQB), even when it does not have data to transmit. If the Card has data (DA = 1), then the Host is responsible for clocking and receiving the entire packet length and inputting all bytes as defined in the packet count. The Host SHALL transmit the IQB immediately following the last byte of the previous packet byte, even if the previous packet did not include any data. This allows for a high-speed interface for both Host and Card sourced data without resorting to separate interrupt signals.

The Host SHALL repeatedly send the IQB followed by 2 bytes of packet count until a message transfer begins. If the Host does not have any data to send, the packet count following the IQB SHALL be set to 0.

The Host and Card both SHALL have separate read and write buffers that are 4,096 bytes, excluding the interface query byte and packet count.

Each packet SHALL only be either a command or extended type, i.e., no mixing of types.

For the command channel, only one SPDU SHALL be allowed per packet.

For the extended channel, only one flow ID SHALL be allowed per packet.

If a packet is not segmented, then both the F and L bits SHALL be set.

Packets smaller than 4,096 bytes SHALL NOT be segmented.

If a packet is larger than 4,096 bytes, then it SHALL be segmented into contiguous packets. The F bit SHALL be set for the first packet. The L bit SHALL be set for the last packet.

The ready flags (HR & CR) SHALL be used for flow control.

The interface is assumed to be a reliable interface. Any error condition detected by the Card (i.e., under-run) SHALL cause the ER bit to be set. This SHOULD be considered to be a catastrophic failure during normal operation causing the Host to reset the Card. It is the Host's responsibility to perform the reset operation. There is an ER bit for the Host to set. However, the Card SHOULD ignore this bit.

### 7.6.3 S-Mode Initialization and Operation

This section defines the interface initialization procedure between the Card operating in S-Mode and the Host.

#### 7.6.3.1 Descriptions

Initialization is a very general term. The following are definitions of how the term initialization is used in this section.

Any computing device SHALL go through an initialization phase whenever a reset condition occurs, such as when initial power is applied, manual reset, or an unrecoverable software error condition occurs. What is covered in this section is the initialization of the interface between the host and the Card. This is defined to be the *interface initialization*.

#### 7.6.3.2 CableCARD Device Personality Change Definition

The host and Card SHALL initialize to the PCMCIA interface and will, at a particular point in the sequence, change to the CableCARD device interface. This point is defined as the *CableCARD Personality Change*.

#### 7.6.3.3 Reset Definition

There are two possible resets that can occur in the Card interface, a hard reset (Called PCMCIA reset) and a soft reset (called Card reset or POD reset).

##### 7.6.3.3.1 PCMCIA Reset

The PCMCIA reset is defined to be one in which the Host SHALL bring the RESET signal to the Card active. The interface SHALL revert to the PCMCIA interface including no longer routing the MPEG data stream through the Card. Obviously this will cause problems to the viewer and should be avoided except in the case that a catastrophic failure has occurred in the Card or in the interface between the Host and the Card.

##### 7.6.3.3.2 Card Reset

The Card reset is defined to be when the Host sets the RS bit in the control register anytime after the CableCARD personality change has occurred. The Host SHALL set the RS bit in both the data channel and extended channel. The Card SHALL detect whether the RS bit has been set in either channel and, if so, SHALL close all open sessions and transport connections and operation SHALL revert to that of just after the CableCARD personality change. This reset SHALL prevent the change of routing of the MPEG data stream, thereby preventing the viewer from noticing any problems unless the video/audio stream being viewed is scrambled. Since the conditional access session is closed, the Card SHALL cease descrambling the data stream until a new session is opened and the appropriate APDU transmitted to the Card.

The Card reset should occur when the Host detects an error in the Card interface or the Card has set the IIR flag (see below).

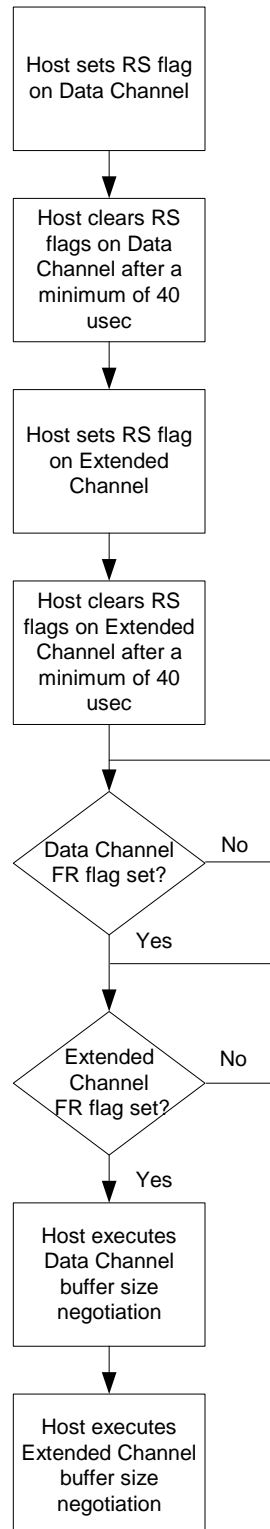


#### **7.6.3.4 Initialize Interface Request Flag**

A status bit called the **Initialize Interface Request (IIR)** flag is included in bit 4 of the status register to allow the Card to request that the interface be re-initialized. This bit exists in both the data channel and extended channel. When a condition occurs that the Card needs to request an interface initialization, it SHALL set both IIR bits. Upon recognition of the IIR flag being set, the Host SHALL implement a Card reset. The Card will clear the IIR flag when the RS bit is set. To further ensure reliable interoperability, the Card SHALL be prohibited from sending LPDUs to the Host after setting the IIR bit and prior to recognizing an active RS bit.

#### **7.6.3.5 Detailed CableCARD Reset Operation**

The following flowchart (Card Reset Sequence) is the required implementation of the Card RS operation. LPDUs SHALL NOT be transmitted until the completion of the Card Reset sequence.

**Figure 7.6-7 - Card RS Operation**

### 7.6.3.6 Configuration Option Register

The Card and the Host SHALL support the Function Configuration as defined in Section 4.14 of [PCMCIA2]. Cards SHALL support only the Configuration Option Register. Host support for registers other than the Configuration Option Register is optional.

The Configuration Option Register (COR) in the Card is only accessible prior to the CableCARD personality change (Section 7.3.5). After the CableCARD personality change, the COR is no longer available. Any relevant configuration data SHALL be transferred via the data or extended channels and is not covered in this document.

By writing the COR with the value defined by the Configuration-Table index byte, TPCE\_INDXX, from the CISTPL\_CFTABLE\_ENTRY tuple, the Host configures the Card into the CableCARD device mode, thus causing the CableCARD personality change.

### 7.6.3.7 Initialization Conditions

There are four possible conditions that can cause the PCMCIA interface initialization phase, they are as follows:

1. The Host and Card are powered up at the same time. After both have performed their internal initialization, then the interface initialization SHALL begin.
2. Host has been powered and in a steady state. A Card is then inserted. After the Card has performed its internal initialization, the interface initialization phase SHALL begin.
3. The Host has performed a reset operation for some reason (spurious signal, brownout, software problem, etc.) that has not caused the Card to reset. The Host SHALL go through its initialization and then SHALL perform a PCMCIA reset on the Card. After the Card has performed its internal initialization, then the interface initialization SHALL begin.
4. The Card has performed a reset operation for some reason (spurious signals, software problem, etc.) that has not caused the Host to reset. The Host SHALL incorporate the timeout detection and will thus detect a timeout and SHALL perform a Card reset.

### 7.6.3.8 OOB Connection and Disconnection Behavior

If a Card is not connected to the Host, the OOB transmitter in the Host SHALL not operate. Upon connection of a Card, the Host SHALL initiate, with the Card, the low-level personality change sequence defined in Section 7.6.3.9 of this document. If successful, the Host SHALL then activate the OOB transmitter as instructed by the Card.

The OOB receiver in the Host SHALL be connected only to the Card interface.

### 7.6.3.9 Low Level Step by Step CableCARD Device Personality Change Sequence

The CableCARD personality change covers the detection of the Card and the transition to the CableCARD device interface. A step-by-step operation for the interface initialization of the physical layer from the Card's viewpoint is defined below.

1. The Card is inserted or already present in a Host.
2. Please refer to section 4.12.1 of [PCMCIA2] for timing diagrams and specifications.

**Power-up:** Power is applied to the Card with the RESET signal in a high-Z state for a minimum of 1 msec after VCC is valid (section 4.4.20 of the PC Card Electrical Specification). The Card's READY signal (pin 16) SHALL be inactive (logic 0) within 10 usec after the RESET signal goes inactive (logic 0), unless the Card will be ready for access within 20 msec after RESET goes inactive. Note that at this time the Card SHALL only operate as an un-configured PCMCIA module.

**PCMCIA Reset:** The RESET signal goes active for a minimum of 10 usec. The Card's READY signal (pin 16) SHALL be inactive (logic 0) within 10 usec after RESET goes inactive (logic 0), unless the Card will be ready for access within 20 msec after RESET goes inactive. Note that at this time the Card SHALL only operate as an un-configured PCMCIA module.

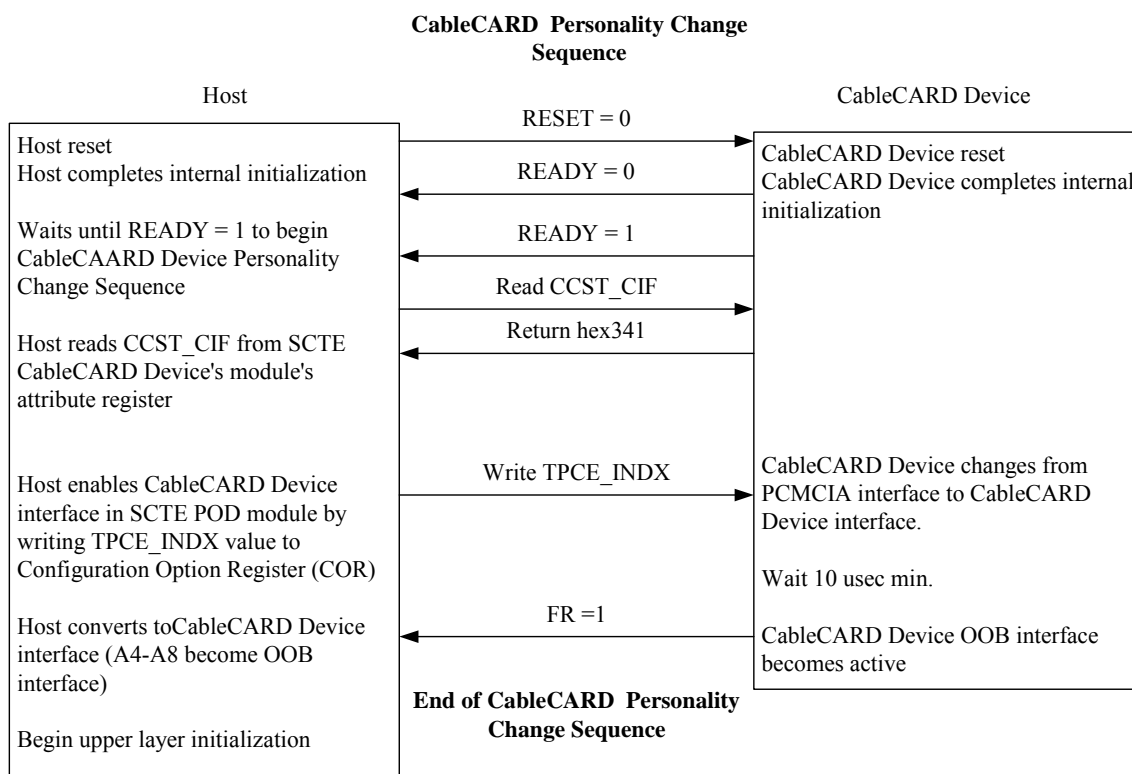
3. After a minimum of 20 msec after RESET goes inactive (section 4.4.6 of [PCMCIA2]), the Host SHALL test

the Card's READY signal. It SHALL NOT attempt to access the Card until the READY signal is active (logic 1).

4. After the Card has completed its PCMCIA internal initialization, it SHALL bring the READY signal active. At this time, all of the interface signals are defined by the PC Card interface standard for Memory Only Card interface (Table 4-1 of [PCMCIA2]). The Card SHALL bring READY active within 5 seconds after RESET goes inactive (section 4.4.6 of [PCMCIA2]).
5. The Host SHALL read the Configuration Information Structure (CIS) available in the attribute memory to determine that the device is a CableCARD device, what version is used, and any other pertinent information. This data is outlined in Section 7.3.5.1 of this document with the revisions in Section 6 of this document.
6. The Host SHALL read all the CCST\_CIF subtuples to verify that the SCTE interface ID number (STCI\_IFN) is present (0x341).  

Informative Note: If it is not present, this means that a different PCMCIA module has been inserted, which is not capable of operating with the SCTE format, however, it may be capable of operating as an NRSS-B module (EIA-679-B Part B).
7. The Host SHALL then write into the COR the value read in TPCE\_INDEX. Following this write cycle, the Host SHALL switch the address signals A4-A8 to the OOB interface signals and the inband transport stream signals. The Host SHALL implement a pull-down resistor on the ETX signal to prevent spurious operation of the transmitter. It SHALL also implement a pull down resistor on the MCLKO signal to prevent invalid inband transport data from being received prior to the CableCARD device personality change.
8. At a maximum of 10 usec after the COR write signal, the Card SHALL switch to the OOB interface signals and the inband transport stream signals.
9. In the event that the Card requires additional initialization, it SHALL NOT bring the FR bit in the status register active until it is ready to begin communications with the Host.
10. This completes the physical link layer initialization.

The following diagram helps define this operation.



**Figure 7.6-8 - CableCARD Personality Change Sequence**

### 7.6.3.10 Initialization Overview

The following sections provide a description of the initialization procedure that SHALL occur between the Card and the Host.

### 7.6.3.11 Physical Layer Initialization

The physical layer initialization covers the buffer size negotiation of both the data and extended channels, and the initialization of the Host-Card transport layer and resource manager. The following physical layer initialization SHALL be implemented in the order listed.

#### 7.6.3.11.1 Data Channel Initialization

The data channel is initialized by the Host writing a '1' to the RS bit in the data channel Control/Status Register. After a minimum of 40 usec, the Host will write a '0' to the RS bit in the data channel Control/Status Register. The Card SHALL clear out any data in the data channel data buffer and configures the Card interface so it can perform the data channel buffer size negotiation protocol. When the Card is ready, it sets the data channel FR bit to '1'.

#### 7.6.3.11.2 Extended Channel Initialization

The extended channel is initialized by the Host writing a '1' to the RS bit in the extended channel Control/Status Register. After a minimum of 40 usec, the Host will write a '0' to the RS bit in the extended channel Control/Status Register. The Card SHALL clear out any data in the extended channel data buffer and configure the Card interface so it can perform the extended data channel buffer size negotiation protocol. When the Card is ready, it sets the extended channel FR bit to '1'.

### **7.6.3.11.3 Data Channel Buffer Size Negotiation**

When a PC Card is plugged into a Host. The PC Card initialization commences with sensing of the Card being plugged in by sense pins on the interface. The Host then reads the Card information Structure residing in the Attribute Memory of the Card. This contains low-level configuration information for the Card, such as PC Card read and write addresses used by the Card, and indicates to the Host that it is a CableCARD device. The Host now turns off the Transport Stream Interface bypass link and allows the transport packets to flow through the Card. This introduces a delay, and consequently a short gap in the Transport Stream data, but this is unavoidable. At the same time the physical layer interface initialization process takes place to negotiate the buffer size to be used for communication. At this point the physical layer initialization process is complete and the upper-layer initialization process, common to all physical implementations, commences with the Host creating a Transport Layer connection to the module.

During Card initialization and at other times if there is an error, the Host needs to be able to reset the interface. It does this by writing a '1' to the RS bit in the Control Register. The Card clears out any data in its data transfer buffer(s) and sets the interface so that it can perform the buffer size negotiation protocol. The Card signals that the Reset operation is complete by setting the FR bit to '1'. After initialization the Host must find out the internal buffer size of the Card by operating the buffer size negotiation protocol. Neither Host nor Card may use the interface for transferring data until this protocol has completed. The Host starts the negotiation by writing a '1' to the SR bit in the Control Register, waiting for the DA bit to be set and then reading the buffer size by a Card to Host transfer operation. At the end of the transfer operation the host resets the SR bit to '0'. The data returned will be 2 bytes with the most significant byte first. The Card SHALL support a minimum buffer size of 16 bytes. The maximum is set by the limitation of the Size Register (65535 bytes). Similarly the Host may have a buffer size limitation that it imposes. The Host SHALL support a minimum buffer size of 256 bytes but it can be up to 65535 bytes. After reading the buffer size the Card can support, the Host takes the lower of its own buffer size and the Card buffer size. This will be the buffer size used for all subsequent data transfers between Host and Card. The Host now tells the Card to use this buffer size by writing a '1' to the SW bit in the Command Register, waiting until the FR bit is set and then writing the size as 2 bytes of data, most significant byte first, using the Host to Card transfer operation. At the end of the transfer the Host sets the SW bit to '0'. The negotiated buffer size applies to both directions of data flow, even in double buffer implementations.

All future data channel transaction buffer sizes SHALL NOT exceed the maximum buffer size. Note that a data channel transaction's buffer size can be smaller than the negotiated buffer size.

### **7.6.3.11.4 Extended Channel Buffer Size Negotiation**

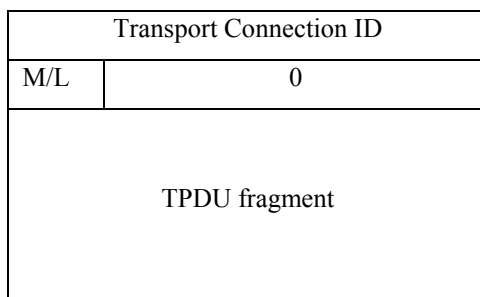
The Extended Channel buffer size negotiation is the same as the data channel defined in Section 7.6.3.11.3. Note that the buffer sizes of the data and extended channels do not have to be the same. The minimum buffer size for the Card is 16 bytes and the minimum buffer size for the Host is 256 bytes. The maximum size for both is 65,535 bytes.

Using the Buffer Size Negotiation protocol called out in Section 7.6.3.11.3, the Host will read the Card's extended channel buffer size, compare the result to its extended channel buffer size, and write the smaller of the two buffer sizes to the Card's extended channel. All future extended channel transaction buffer sizes SHALL NOT exceed the maximum buffer size. Note that the extended channel transaction's buffer size can be smaller than the negotiated buffer size.

### **7.6.3.11.5 Link Connection**

The Link Layer on the Command Interface does two jobs. It fragments Transport Protocol Data Units (TPDU), if necessary, for sending over the limited buffer size of the Physical Layer, and reassembles received fragments. It also fairly multiplexes several Transport Connections onto the one Link Connection. It does this by interleaving fragments from all the Transport Connections which are currently trying to send TPDU's over the link. It assumes that the Physical Layer transfer mechanism is reliable, that is, it keeps the data in the correct order and neither deletes nor repeats any of it.

A Link Connection is established automatically as a consequence of the establishment of the Physical Layer connection, that is, plugging in the Card or powering up, reading the Card Information Structure, and configuring the Card in the appropriate mode. No further explicit establishment procedure is required. The size of each Link Protocol Data Unit (LPDU) depends on the size that the Host and Card negotiated using the SR & SW commands on the interface. Each LPDU consists of a two-byte header followed by a fragment of a TPDU, the total size not exceeding the negotiated buffer size. The first byte of the header is the Transport Connection Identifier for that TPDU fragment. The second byte contains a More/Last indicator in its most significant bit. If the bit is set to '1' then at least one more TPDU fragment follows, and if the bit is set to '0' then it indicates this is the last (or only) fragment of the TPDU for that Transport Connection. All other bits in the second byte are reserved and SHALL be set to zero. This is illustrated in Figure 7.6-9.



**Figure 7.6-9 - Layout of Link Protocol Data Unit**

Each TPDU SHALL start in a new LPDU, that is, the LPDU carrying the last fragment of the previous TPDU on a Transport Connection cannot also carry the first fragment of the next one. If more than one Transport Connection currently has TPDUs in transit the Link Layer SHALL send a fragment for each of them in turn, so that all Transport Connections get a fair apportionment of the communication bandwidth available.

No explicit initialization of the Link Layer is required.

#### **7.6.3.11.6 Host-CableCARD Device Transport Layer Connection**

The transport layer (TPDU) connection is covered in the following sections, and SHALL be supported with the addition of the following: “TPDU chaining SHALL NOT be supported. The maximum length of the transport data SHALL be limited to 65,534 bytes.”

The communication of data across this Command Channel is defined in terms of objects plus the interrupt mode extension. Card applications SHALL use this path when they require support from Host resources. The objects are coded by means of a general Tag-Length-Value coding derived from that used to code ASN.1 syntax.

**Table 7.6–5 - Length field used by all PDUs at Transport, Session and Application Layers**

Syntax	# of bits	Mnemonic
Length_field(){ size_indicator if (size_indicator ==0) length_value else if (size_indicator==1){ length_field_size for (i=0;i<length_field_size; i++){ length_value_byte } } }	1	bslbf
	7	uimsbf
	7	uimsbf
	8	bslbf

This section describes the ASN.1 objects for the Transport and Session Layers that travel over the command interface. For all these objects, and for the Application Layer objects, the coding in Table 7.6–5 applies for the Length field, which indicates the number of bytes in the following Value field.

Size\_indicator is the first bit of the length\_field. If size\_indicator = 0, the length of the data field is coded in the succeeding 7 bits. Any length from 0 to 127 can thus be encoded on one byte. If the length exceeds 127, then size\_indicator is set to 1. In this case, the succeeding 7 bits code the number of subsequent bytes in the length field. Those subsequent bytes shall be concatenated, first byte at the most significant end, to encode an integer value. Any value field length up to 65535 can thus be encoded by three bytes. The indefinite length format specified by the basic encoding rules of ASN.1 is not used [ISO8825].

#### 7.6.3.11.6.1 Transport Layer

The Transport Layer of the Command Channel operates on top of a Link Layer provided by the particular physical implementation used. The transport protocol assumes that the Link Layer is reliable, that is, data is conveyed in the correct order and with no deletion or repetition of data. The transport protocol is a command-response protocol where the Host sends a command to the Card, using a Command Transport Protocol Data Unit (C\_TPDU) and waits for a response from the module with a Response Transport Protocol Data Unit (R\_TPDU). The Card cannot initiate communication: it must wait for the Host to poll it or send it data first. The protocol is supported by Transport Layer objects. Some of them appear only in C\_TPDUs from the Host, some only in R\_TPDUs from the Card and some can appear in either. Create\_T\_C and C\_T\_C\_Reply, create new Transport Connections. Delete\_T\_C and D\_T\_C\_Reply, shut them down. Request\_T\_C and New\_T\_C allow a Card to request the Host to create a new Transport Connection. T\_C\_Error allows error conditions to be signaled. T\_SB carries status information from module to Host. T\_RCV requests waiting data from a Card and T\_Data\_More and T\_Data\_Last convey data from higher layers between Host and Card. T\_Data\_Last with an empty data field is used by the Host to poll regularly for data from the Card when it has nothing to send itself. A C\_TPDU from the Host contains only one Transport Protocol Object. A R\_TPDU from a Card may carry one or two Transport Protocol Objects. The sole object or second object of a pair in a R\_TPDU is always a T\_SB object.

#### 7.6.3.11.6.2 Transport protocol objects

All transport layer objects contain a transport connection identifier. This is one octet, allowing up to 255 Transport Layer connections to be active on the Host simultaneously. Transport connection identifier value 0 is reserved. The identifier value is always assigned by the Host. The protocol is described in detail here as it is common to all physical implementations but the objects are only described in general terms. The detailed coding of the objects depends upon the particular physical layer used.

1. Create\_T\_C creates the Transport Connection. It is only issued by the Host and carries the transport connection identifier value for the connection to be established.
2. C\_T\_C\_Reply is the response from the target module to Create\_T\_C and carries the transport connection identifier for the created connection.
3. Delete\_T\_C deletes an existing Transport Connection. It has as a parameter the transport connection identifier for the connection to be deleted. It can be issued by either Host or Card. If issued by the Card it does so in response to a poll or data from the Host.
4. D\_T\_C\_Reply is the reply to the delete. In some circumstances this reply may not reach its destination, so the Delete\_T\_C object has a time-out associated with it. If the time-out matures before the reply is received then all actions which would have been taken on receipt of the reply can be taken at the timeout.
5. Request\_T\_C requests the Host to create a new Transport Connection. It is sent on an existing Transport Connection from that Card. It is sent in response to a poll or data from the Host.
6. New\_T\_C is the response to Request\_T\_C. It is sent on the same Transport Connection as the Request\_T\_C object, and carries the transport connection identifier of the new connection. New\_T\_C is immediately followed by a Create\_T\_C object for the new connection, which sets up the Transport Connection proper.



7. T\_C\_Error is sent to signal an error condition and carries a 1-byte error code specifying the error. This is sent in response to Request\_T\_C to signal that no more Transport Connections are available.
8. T\_SB is sent as a reply to all objects from the Host, either appended to other protocol objects or sent on its own, as appropriate. It carries one byte which indicates if the module has data available to send.
9. T\_RCV is sent by the Host to request that data the module wishes to send (signaled in a previous T\_SB from the Card) be returned to the Host.
10. T\_Data\_More and T\_Data\_Last convey data between Host and Card, and can be in either a C\_TPDU or a R\_TPDU. From the Card they are only ever sent in response to an explicit request by a T\_RCV from the Host. T\_Data\_More is used if a Protocol Data Unit (PDU) from a higher layer has to be split into fragments for sending due to external constraints on the size of data transfers. It indicates that at least one more fragment of the upper-layer PDU will be sent after this one. T\_Data\_Last indicates the last or only fragment of an upper-layer PDU.

#### 7.6.3.11.6.3 Transport protocol

When the Host wishes to set up a transport connection to a Card, it sends the Create\_T\_C object and moves to state 'In Creation'. The Card shall reply directly with a C\_T\_C\_Reply object. If after a time-out period the Card does not respond, then the Host returns to the idle state (via the 'Time-out' arc). The Host will not transmit or poll again on that particular transport connection, and a late C\_T\_C\_Reply will be ignored. If, subsequently, the Host re-uses the same transport connection identifier, then the Card will receive Create\_T\_C again, and from this it SHALL infer that the old transport connection is dead, and a new one is being set up.

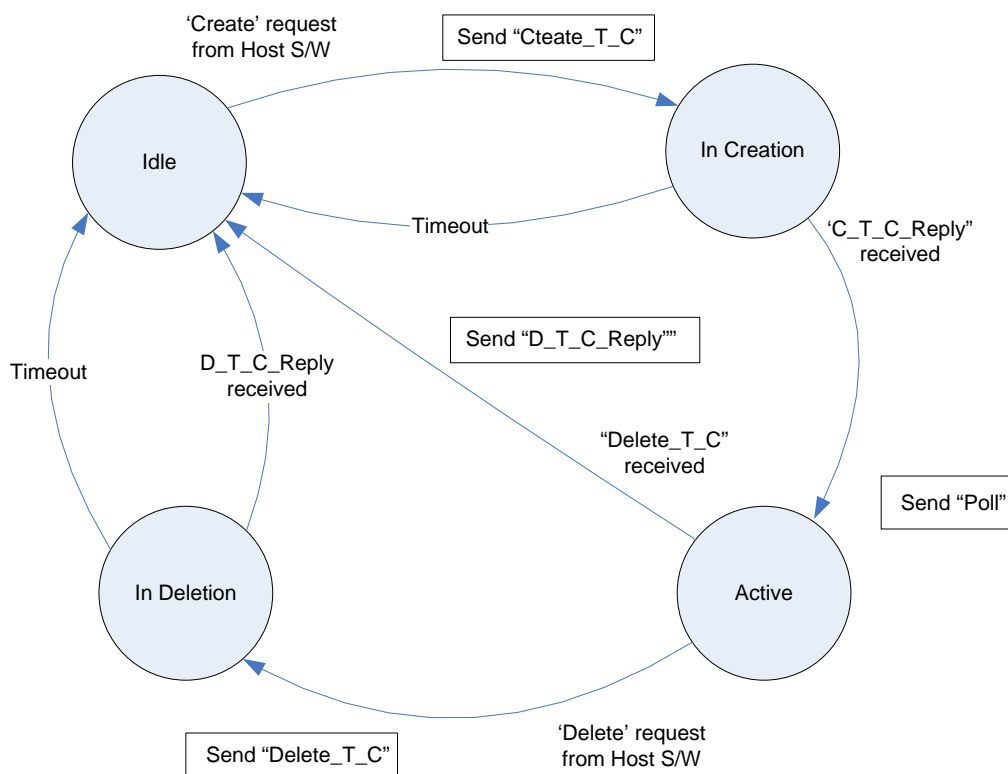
When the Card replies with C\_T\_C\_Reply, the Host moves to the 'Active' state of the connection. If the Host has data to send, it can now do so, but otherwise it issues a poll and then polls regularly thereafter on the connection.

If the Host wishes to terminate the transport connection, it sends a Delete\_T\_C object and moves to the 'In Deletion' state. It then returns to the 'Idle' state upon receipt of a D\_T\_C\_Reply object, or after a time-out if none is received. If the Host receives a Delete\_T\_C object from the module it issues a D\_T\_C\_Reply object and goes directly to the idle state. Except for the 'Active' state, any object received in any state which is not expected is ignored.

In the 'Active' state the Host issues polls periodically, or sends data if it has an upper-layer PDU to send. In response it receives a T\_SB object, preceded by a Request\_T\_C or Delete\_T\_C object if that is what the Card wants to do.

In the 'Active' state, data can be sent by the Host at any time. If the Card wishes to send data it must wait

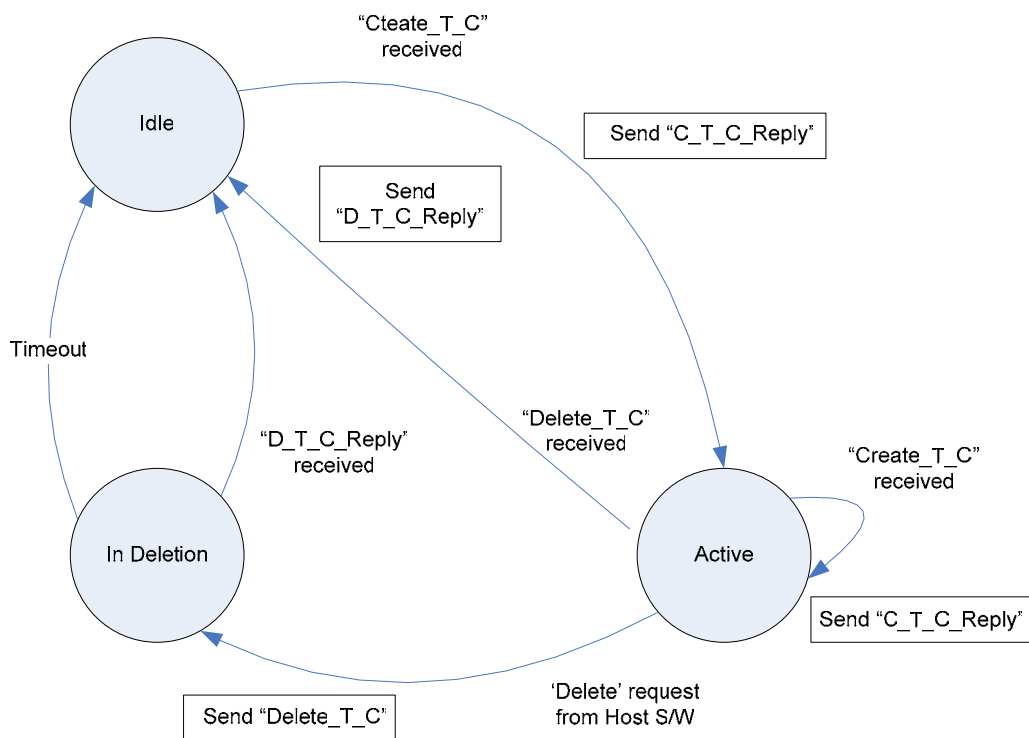
for a message from the Host - normally data or a poll - and then indicate that it has data available in the T\_SB reply. The Host will then at some point - not necessarily immediately - send a T\_RCV request to the Card to which the Card responds by sending the waiting data in a T\_Data object. Where T\_Data\_More is used, each subsequent fragment must wait for another T\_RCV from the Host before it can be sent.



**Figure 7.6-10 - State Transition Diagram – Host Side of the Transport Protocol**

**Table 7.6–6 - Expected Received Objects – Transport Connection on the Host**

State	Expected Objects - Host
Idle	None
In Creation	C_T_C_Reply (+T_SB)
Active	T_Data_More, T_Data_Last, Request_T_C, Delete_T_C, T_SB
In Deletion	D_T_C_Reply (+T_SB)



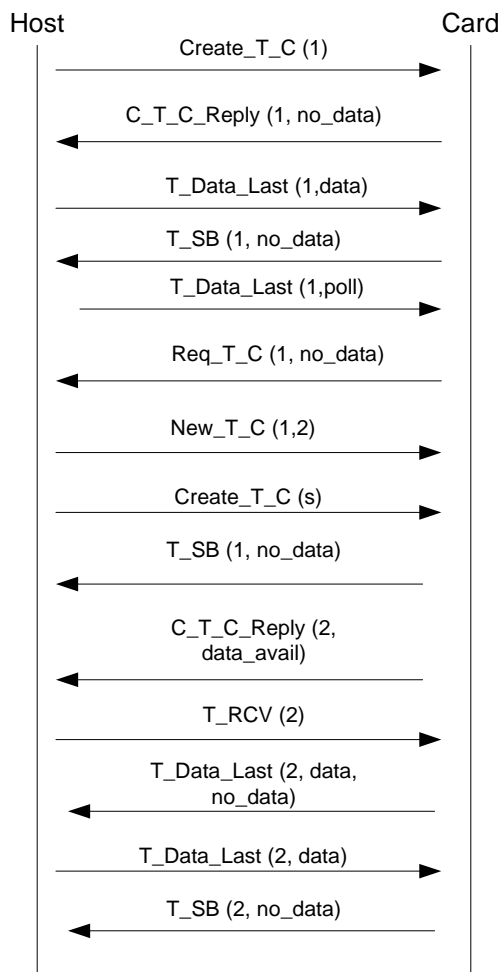
**Figure 7.6-11 - State Transition Diagram – Card Side of the Transport Protocol**

**Table 7.6–7 - Expected Received Objects – Transport Connection on the Card**

State	Expected Objects - Host
Idle	Create_T_C
Active	Create_T_C, T_Data_More, T_Data_Last, New_T_C, Delete_T_C, T_RCV, T_C_Error
In Deletion	D_T_C_Reply

If a Card wishes to set up another Transport Connection it SHALL send a Request\_T\_C object either in response to a poll or data. If it can meet the request the Host will reply with a New\_T\_C containing the transport connection identifier for the new connection, immediately followed by a Create\_T\_C object to create the connection. If the Host cannot meet the request because all transport connection identifiers are in use then it SHALL reply with a T\_C\_Error containing the appropriate error code.

An example of an object transfer sequence to create and use a Transport Connection is illustrated in Figure 7.6-12.



**Figure 7.6-12 - Object Transfer Sequence – Transport Protocol**

In this example it is assumed that the Card has just been plugged in and a physical connection has been established (PC Card initialization, etc.). The Host now issues a `Create_T_C` for Transport Connection number 1. The Card replies immediately with `C_T_C_Reply` for Transport Connection 1, also indicating it has no data to send. The Host now sends some data with a `T_Data_Last` and the Card responds with just a `T_SB` indicating no data to send. Some time later the Host polls the Card with an empty `T_Data_Last`, and the Card responds with a `Request_T_C` saying it wishes to have a new Transport Connection created, and also indicating (in the appended `T_SB`) that it has no data to send on connection 1. The Host replies with `New_T_C` indicating that Transport Connection 2 will be set up. This is immediately followed by `Create_T_C` for Transport Connection 2. The Card responds with a `T_SB` to the first and a `C_T_C_Reply` to the second, also indicating that it has data to send on this connection. The Host responds with `T_RCV` to receive the data, and the Card responds with `T_Data_Last` containing the data. The Host replies with data of its own and the Card responds to that indicating it has no further data to send. Both Transport Connections now persist until either Card or Host deletes one, and the Host polls periodically on both connections with an empty `T_Data_Last`.

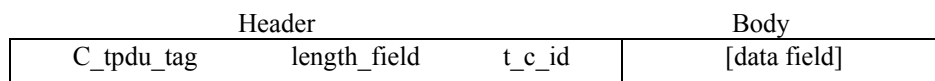
#### 7.6.3.11.6.4 Transport Protocol Objects

The transport protocol on the Command Channel is a command-response protocol where the Host sends a command to the Card, using a Command Transport Protocol Data Unit (`C_TPDU`) and waits for a response from the Card with a Response Transport Protocol Data Unit (`R_TPDU`). The Card cannot initiate communication: it must wait for the Host to poll it or send it data first. The protocol is supported by eleven Transport Layer objects. Some of them appear only in `C_TPDUs` from the Host, some only in `R_TPDUs` from the Card and some can appear in either. `Create_T_C` and `C_T_C_Reply`, create new Transport Connections. `Delete_T_C` and `D_T_C_Reply`, clear them

down. Request\_T\_C and New\_T\_C allow a Card to request the Host to create a new Transport Connection. T\_C\_Error allows error conditions to be signaled. T\_SB carries status information from the Card to the Host. T\_RCV requests waiting data from a Card and T\_Data\_More and T\_Data\_Last convey data from higher layers between the Host and the Card. T\_Data\_Last with an empty data field is used by the Host to poll regularly for data from the Card when it has nothing to send itself. In all objects there is a tag\_field and a length\_field coded according to the rules defined in [ISO8825], and a t\_c\_id field which is a single octet.

#### 7.6.3.11.6.4.1 Command TPDU

The Command TPDU (C\_TPDU) conveys Transport Protocol objects from Host to Card.



**Figure 7.6-13 - C\_TPDU Structure**

**Table 7.6–8 - Command TPDU (C\_TPDU)**

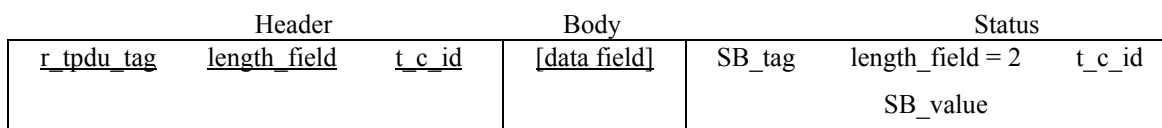
Syntax	No. of Bits	Mnemonic
C_TPDU() {		
c_tpdu_tag	8	uimsbf
length_field()		
t_c_id	8	uimsbf
for (i=0; i<length_value; i++) {		
data_byte	8	uimsbf
}		
}		

The C\_TPDU is made of two parts:

1. A mandatory header made of a tag value c\_tpdu\_tag, coding the TPDU command, a length\_field, coding the length of all the following fields, and a transport connection identifier noted t\_c\_id.
2. A conditional body of variable length equal to the length coded by length\_field minus one.

#### 7.6.3.11.6.4.2 Response TPDU

The Response TPDU (R\_TPDU) conveys Transport Protocol objects from Card to Host.



**Figure 7.6-14 - R\_TPDU Structure**

**Table 7.6–9 - Response TPDU (R\_TPDU)**

Syntax	No. of Bits	Mnemonic
R_TPDU() {		
r_tpdu_tag	8	uimsbf
length_field()		
t_c_id	8	uimsbf
for (i=0; i<length_value; i++) {		
data_byte	8	uimsbf
}		
SB_tag	8	uimsbf
length_field()=2		
t_c_id	8	uimsbf
SB_value	8	uimsbf
}		

The R\_TPDU is made of three parts:

1. A conditional header made of a tag value r\_tpdu\_tag, coding the TPDU response, a length field, coding the length of the following transport connection identifier and data fields, and a transport connection identifier field noted t\_c\_id. The status is not included in the calculation of length\_field.
2. A conditional body of variable length equal to the length coded by length\_field minus one.
3. A mandatory Status made of a Status tag SB\_tag, a length\_field equal to 2, a transport connection identifier and a one-byte Status Byte value (SB\_value) coded according to Figure 7.6-15 and Table 7.6–10.

Bit	7	6	5	4	3	2	1	0
	DA	reserved						

**Figure 7.6-15 - SB\_value**

The 1-bit DA (Data Available) indicator field indicates whether the module has a message available in its output buffer for the host. The host has to issue a Receive\_data C\_TPDU to get the message (see 7.6.3.11.6.5). The coding of DA indicator is given in Table 7.6–10. The ‘reserved’ field shall be set to zero.

**Table 7.6–10 - Coding of bit8 of SB\_value**

Bit 7	Meaning
0	No message available
1	Message available

#### 7.6.3.11.6.4.3 Create Transport Connection (Create\_T\_C)

The Host SHALL request to open a transport connection. The host SHALL open exactly one transport connection for each Card.

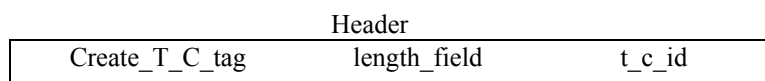
**Table 7.6–11 - Create Transport Connection (Create\_T\_C)**

Syntax	Value (hex)	# of bits	Mnemonic
Create_T_C() {			
create_T_C_tag	0x82	8	uimsbf
length_field()	0x01	8	uimsbf
t_c_id	XX	8	uimsbf
}			

Where XX is defined by the Host. A transport connection ID (t\_c\_ID) value of zero is invalid.

The Create\_T\_C object is made of only one part:

A mandatory header made of a tag value create\_T\_C\_tag, coding the Create\_T\_C object, a length\_field equal to one, and a transport connection identifier noted t\_c\_id.

**Figure 7.6-16 - Create\_T\_C Structure**

#### 7.6.3.11.6.4.4 Create Transport Connection Reply (C\_T\_C\_Reply)

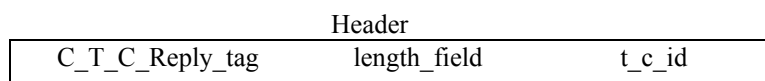
The Card SHALL respond with the following.

**Table 7.6–12 - Create Transport Connection Reply (C\_T\_C\_Reply)**

Syntax	Value (hex)	# of bits	Mnemonic
C_T_C_Reply () {			
C_T_C_Reply_tag	0x83	8	uimsbf
length_field()	0x01	8	uimsbf
t_c_id	XX	8	uimsbf
}			

The C\_T\_C\_Reply object is made of only one part:

A mandatory header made of a tag value C\_T\_C\_Reply\_tag, coding the C\_T\_C\_Reply object, a length\_field equal to one, and a transport connection identifier.

**Figure 7.6-17 - C\_T\_C\_Reply Structure**

A transport connection of ID “XX” now exists.

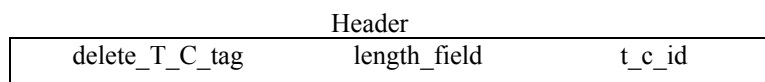
#### 7.6.3.11.6.4.5 Delete Transport Connection (Delete\_T\_C)

**Table 7.6–13 - Delete Transport Connection (Delete\_T\_C)**

Syntax	Value (hex)	# of bits	Mnemonic
delete_T_C () {			
C_T_C_Reply_tag	0x84	8	uimsbf
length_field()	0x01	8	uimsbf
t_c_id	XX	8	uimsbf
}			

The Delete\_T\_C object is made of only one part:

A mandatory header made of a tag value delete\_T\_C\_tag, coding the Delete\_T\_C object, a length\_field equal to one, and a transport connection identifier.



**Figure 7.6-18 - Delete\_T\_C Structure**

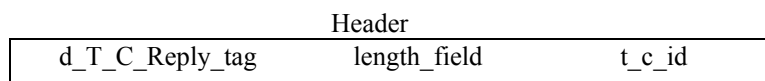
#### 7.6.3.11.6.4.6 Delete Transport Connection Reply (D\_T\_C\_Reply)

**Table 7.6–14 - Delete Transport Connection Reply (D\_T\_C Reply)**

Syntax	Value (hex)	# of bits	Mnemonic
D_T_C_Reply () {			
d_T_C_Reply_tag	0x85	8	uimsbf
length_field()	0x01	8	uimsbf
t_c_id	XX	8	uimsbf
}			

The D\_T\_C\_Reply object is made of only one part:

A mandatory header made of a tag value d\_T\_C\_Reply\_tag, coding the D\_T\_C\_Reply object, a length\_field equal to one, and a transport connection identifier.



**Figure 7.6-19 - D\_T\_C\_Reply Structure**

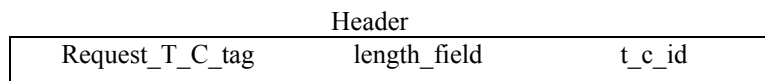
#### 7.6.3.11.6.4.7 Request Transport Connection (Request\_T\_C)

**Table 7.6–15 - Request Transport Connection (Request\_T\_C)**

Syntax	Value (hex)	# of bits	Mnemonic
Request_T_C () {			
Request_T_C_tag	0x86	8	uimsbf
length_field()	0x01	8	uimsbf
t_c_id	XX	8	uimsbf
}			

The Request\_T\_C object is made of only one part:

A mandatory header made of a tag value request\_T\_C\_tag, coding the Request\_T\_C object, a length\_field equal to one, and a transport connection identifier.



**Figure 7.6-20 - Request\_T\_C Structure**



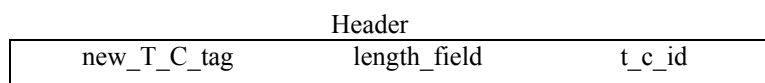
## 7.6.3.11.6.4.8 New Transport Connection (New\_T\_C)

**Table 7.6–16 - New Transport Connection (New\_T\_C)**

Syntax	Value (hex)	# of bits	Mnemonic
New_T_C () {			
new_T_C_tag	0x87	8	uimsbf
length_field()	0x02	8	uimsbf
t_c_id	XX	8	uimsbf
new_t_c_id	XX	8	uimsbf
}			

The New\_T\_C object is made of two parts:

1. A mandatory header made of a tag value new\_T\_C\_tag, coding the New\_T\_C object, a length\_field equal to two and a transport connection identifier.
2. A mandatory body consisting of the transport connection identifier for the new connection to be established.

**Figure 7.6-21 - Request\_T\_C Structure**

## 7.6.3.11.6.4.9 Transport Connection Error (T\_C\_Error)

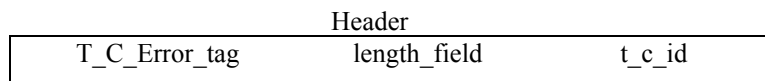
**Table 7.6–17 - Transport Connection Error (T\_C\_Error)**

Syntax	Value (hex)	# of bits	Mnemonic
T_C_Error () {			
T_C_Error_tag	0x88	8	uimsbf
length_field()	0x02	8	uimsbf
t_c_id	XX	8	uimsbf
error_code	XX	8	uimsbf
}			

The T\_C\_Error object is made of two parts:

1. A mandatory header made of a tag value T\_C\_Error\_tag, coding the T\_C\_Error object, a length\_field equal to two and a transport connection identifier.
2. A mandatory body consisting of the error code for the particular error being signaled.

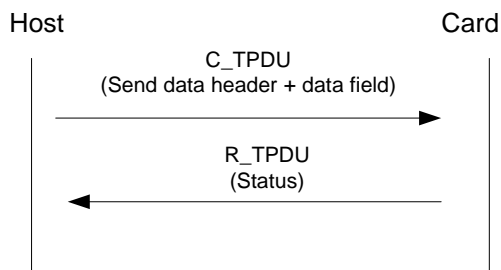
Transport Connection Error Codes are defined in Table 7.6–18.

**Figure 7.6-22 - T\_C\_Error Structure****Table 7.6–18 - Error Code Values**

Error Code	Meaning
1	No transport connections available

## 7.6.3.11.6.5 C\_TPDUs and Associated R\_TPDUs

The send data command is used by the Host, when the transport connection is open, either to send data to the Card or to get information given by the status byte. The Card replies with the status byte. See Figure 7.6-23.



**Figure 7.6-23 - Send Data command/ Response Pair**

**Table 7.6–19 - Send Data C\_TPDU**

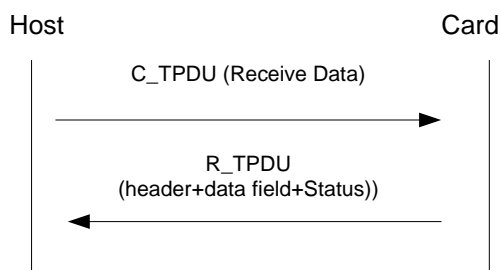
c_TPDU_tag	M_c_TPDU_tag : Tdata_more L_c_TPDU_tag : Tdata_last
length_field	Length of data field according to [ISO8825]
t_c_id	Transport connection identifier
data_field	Subset of (TLV TLV ... TLV

**Table 7.6–20 - Send Data R\_TPDU**

Status	according to Figure 7.6-15
--------	----------------------------

A SEND DATA C\_TPDU with L=1 (no data field) can be issued by the Host just to get information given by the status byte (see polling function, Section 7.6.3.11.6.6).

The receive data command is used by the Host to receive data from the Card. See Figure 7.6-24.



**Figure 7.6-24 - Receive Data command/ Response Pair**

**Table 7.6–21 - Receive Data C\_TPDU**

c_TPDU_tag	T <sub>RCV</sub>
length_field	c_TPDU_length is set to '1'
t_c_id	Transport connection identifier

**Table 7.6–22 - Receive Data R\_TPDU**

r_TPDU_tag	M_r_TPDU_tag : <sup>T</sup> data_more L_r_TPDU_tag : <sup>T</sup> data_last
length_field	Length of data field according to [ISO8825]
t_c_id	Transport connection identifier
data_field	Subset of (TLV TLV ... TLV)
Status	According to Figure 7.6-15

#### 7.6.3.11.6.6 Polling Function Rules

The polling function consists in sending a command to the Card in order to know whether it has data to send to the Host or not.

This function is provided by the Host which has regularly to issue a SEND DATA C\_TPDU with length L equal to 1 (t\_c\_id field only, no data field). As long as data is not available, the Card replies with the status byte having the DA indicator set to 0. When data is available, the Card replies with the status byte having DA indicator set to 1. The Host can then issue one or several RECEIVE DATA C\_TPDU, that will provoke transmission of data, until the DA indicator of the status byte is set to 0 again. While the Card still has data to send, that is, the Host has received status bytes with the DA indicator set, then the poll function shall be suspended. It SHALL be restarted when a status byte is received with the DA indicator not set.

The maximum period of the polling function is equal to 100ms.

At each poll a time-out of 300ms is started, and is reset when the poll response is received. If no poll response has been received within that time, then the transport connection is deleted by the Host in the normal way. The Host does not send any additional polls while waiting for a poll response, even if its normal poll interval is exceeded.

#### 7.6.3.11.6.7 Transport Tag List

**Table 7.6–23 - Transport Tag Values**

tpdu_tag	Tag Value (hex)	Primitive or Constructed	Direction Host ↔ Card
<sup>T</sup> SB	'80'	P	←
<sup>T</sup> RCV	'81'	P	→
<sup>T</sup> create_t_c	'82'	P	→
<sup>T</sup> c_t_c_reply	'83'	P	←
<sup>T</sup> delete_t_c	'84'	P	↔
<sup>T</sup> d_t_c_reply	'85'	P	↔
<sup>T</sup> request_t_c	'86'	P	←
<sup>T</sup> new_t_c	'87'	P	→
<sup>T</sup> t_c_error	'88'	P	→
<sup>T</sup> data_last	'A0'	C	↔
<sup>T</sup> data_more	'A1'	C	↔

#### 7.6.3.11.7 Resource Manager Session Initialization

The Session Layer provides the mechanism by which applications communicate with and make use of resources. The resource is a mechanism for encapsulating functionality at the Application Layer and is described fully in Section 9.1.

Resources vary in the number of simultaneous sessions they can support. Some resources support only one. If a second application tries to request a session to such a resource already in use then it will receive a 'resource busy' reply. Other resources can support more than one simultaneous session, in which case resource requests will be honored up to some limit defined by the resource. An example of the latter would be the display resource, which in some Host implementations may be able to support simultaneous displays in different windows.

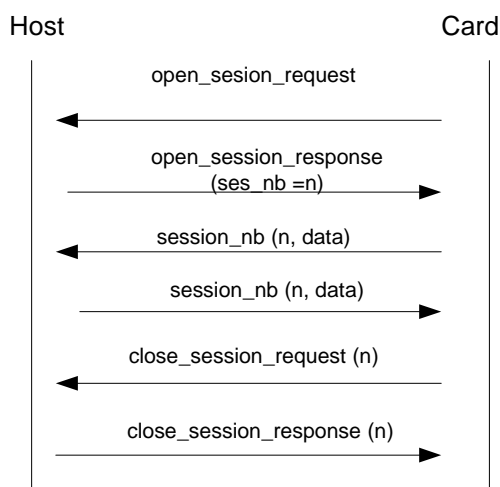
### 7.6.3.11.7.1 Session Protocol Objects

The session objects are described in general terms here, as is the protocol, but the detailed coding of the objects is described in later sections.

1. `open_session_request` is issued by an application over its transport connection to the Host requesting the use of a resource.
2. `open_session_response` is returned by the Host to an application that requested a resource in order to allocate a session number or to tell the Card that its request could not be fulfilled.
3. `close_session_request` is issued by a Card or by the Host to close a session.
4. `close_session_response` is issued by a Card or by the Host to acknowledge the closing of the session.
5. `session_number` always precedes the body of the SPDU containing APDU(s).

### 7.6.3.11.7.2 Session Protocol

The dialogue in the session layer is initiated by a module or by the host. One example is illustrated. In Figure 7.6-25, a Card A wishes to use a resource which is provided by the host.



**Figure 7.6-25 - Object Transfer Sequence – Transport Protocol**

The Card requests a session to be opened to a resource on its transport connection. Since the Host provides the resource itself it replies directly with a session number in its open session response. Communication now proceeds with application layer data preceded by session\_number objects. Eventually the session is closed, in this example, by the Card but it could also have been closed by the Host, for example if the resource became unavailable for any reason. Section 9.1 defines the message structure.

### 7.6.3.11.8 Card Resource Profile

#### Resource Manager Protocol

When a Card is plugged in or the Host is powered up one transport connection is created to the Card, serving an application and/or a resource provider. The first thing an application or resource provider does is to request a session to the Resource Manager resource, which is invariably created since the Resource Manager has no session limit. The Resource Manager then sends a Profile Inquiry to the application or resource provider which responds with a Profile Reply listing the resources it provides (if any). The application or resource provider must now wait for a Profile Change object. While waiting for Profile Change it can neither create sessions to other resources nor can it accept sessions from other applications, returning a reply of ‘resource non-existent’ or ‘resource exists but unavailable’ as appropriate.

When it has asked for profiles on all transport connections and received Profile Replies the Host builds a list of available resources. Where two or more resources match in both class and type the Host keeps the one with the highest version number in its list. Where the version numbers match also the host keeps all resources and chooses one at random when a create session request is received for it. Once the Host has built its resource list it sends a Profile Change object on all current Resource Manager sessions, and those applications that wish to can then ask the Host for its list of resources using the Profile Inquiry object.

When it receives the Profile Change notification for the first time the application or resource provider can interrogate the Host with a Profile Inquiry and receive a Profile Reply with the Host's list of available resources. After this first operation of the Profile Change protocol the application or resource provider is now free to create or accept other sessions. Its session to the Resource Manager persists to allow further Profile Change notification by the Host from time to time.

If a resource provider wishes to notify a change in the profile of resources it provides, it issues a Profile Change to the Host. The Host replies with a Profile Inquiry to which the resource provider replies in turn with its updated resource list. The Host processes this and, if this results in any change to the Host's own resource list, the Host will issue a Profile Change on all active Resource Manager sessions. The applications can then inquire and receive an updated resource list if they wish.

The Card resource profile is obtained by the Host and is covered in Section 9.4. Since the Card is designed to be the only Card in a Host, it SHALL NOT report any resources to the Host.

#### 7.6.3.11.9 Host Resource Profile

The Host SHALL send a *profile\_changed()* APDU so that the Card SHALL then perform a *profile\_inq()* APDU to which the Host SHALL respond with its *profile\_reply()* APDU.

The Host sends:

**Table 7.6–24 - Profile Changed**

Syntax	Value (hex)	# of bits	Mnemonic
<code>profile_changed(){   profile_changed_tag   length_field() }</code>	0x9F8012 00	24 8	uimsbf uimsbf

To which the Card replies with:

**Table 7.6–25 - Profile Inquiry**

Syntax	Value (hex)	# of bits	Mnemonic
<code>profile_inq(){   profile_inq_tag   length_field() }</code>	0x9F8010 00	24 8	uimsbf uimsbf

To which the Host SHALL reply with:

**Table 7.6–26 - Profile Reply**

Syntax	Value (hex)	# of bits	Mnemonic
profile_reply(){			
profile_reply_tag	0x9F8011	24	uimbsf
length_field()	N*4	8	uimbsf
for(i=0; i<N; i++) {			
resource_identifier()	XXXXXXXX	32	uimbsf
}			
}			

Where N is the number of resource identifiers and XXXXXXXX is each unique resource identifier.

NOTE: If a Host supports multiple types of a given resource, each type of that resource will be reported as a resource identifier.

Now the Card knows what resources are available in the Host.

#### **7.6.3.11.10 Application Info Session Initialization**

The Card application SHALL request to open only one session to the Application Information resource to pass application information and to manage application menu entry points. Once the session is created, the Host sends an *application\_info\_req()* APDU to the Card. The Card will respond with the *application\_info\_cnf()* APDU. Detailed operation of the application info is covered in Section 9.2 of this document.

#### **7.6.3.11.11 Conditional Access Application Initialization**

A Conditional Access application in the Card SHALL request to open a single session to the CA Support resource in the Host to allow CA information from the SI and information about user-selected services to be given to the application. Once the session is created, the Host sends a *ca\_info\_inquiry()* APDU to the application, which responds with *ca\_info()* APDU. The Host may then enter into a subsequent dialogue with the CA application to determine which selected services the CA application can descramble and under what conditions. This is described in Section 9.7 of this document. Under normal operating conditions, this session will never be closed.

#### **7.6.3.12 Copy Protection**

A Copy Protection application in the Card SHALL create a session to the Copy Protection resource in the Host. Initialization of Copy Protection is covered in OpenCable CableCARD Copy Protection 2.0 [CCCP2].

#### **7.6.3.13 Extended Channel**

An extended channel application in the Card SHALL create a session to the Extended Channel Support resource to allow for the establishment of flows on the extended channel. These flows will be used for transferring IP packets and MPEG table sections and DSG PDUs across the CHI. Under normal operating conditions, this session will never be closed. Please refer to Section 10 of this document.

#### **7.6.3.14 Host Control**

A Host Control application SHALL create a session to the Host Control resource to allow the Card to control various Host devices. Please refer to Section 9.8 and 10 of this document for details on the Host Control resource.

#### **7.6.3.15 Low Speed Communication**

If reported by the Host as an available resource and the Card implements a Low Speed Communication application, the Card application SHALL create a session to the Low Speed Communication resource to allow the Card to identify what type of FDC, RDC, and any type of Host modem implementations available in the Host.

### 7.6.3.16 Generic IPPV Support

If reported by the Host as an available resource and the Card implements a Generic IPPV application, the Card application SHALL create a session to the Generic IPPV resource to allow the Host to receive information on and to purchase IPPV events. The IPPV Operation is defined in Section 9.9 of this document.

### 7.6.3.17 System Time

The Card SHALL open a single session to the System Time resource to allow the Card to receive system time from the Host.

### 7.6.3.18 Homing

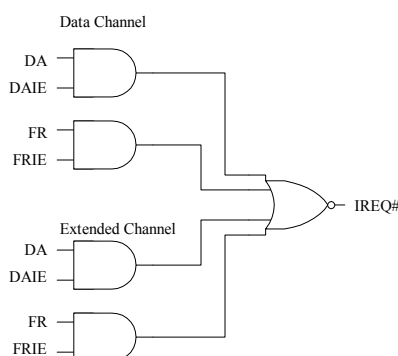
The Card SHALL open a single session to the Homing resource in the Host to allow the Card to determine when it may take control of the tuner. The Homing operation is defined in Section 9.18 of this document.

### 7.6.3.19 Interrupt Operation

Section 7.6.1.1 of this document defines that the PCMCIA IREQ# signal is available for use by the Host. This signal can be utilized by the Host to simplify the physical layer operation but currently cannot be used for transport layer operation.

### 7.6.3.20 Physical Level

The following diagram shows the Card interrupt logical operation.



**Figure 7.6-26 - CableCARD Device Interrupt Logical Operation**

As illustrated in Figure 7.6-26, an interrupt SHALL occur whenever the DA or FR bits are set for either the data or extended channels and their corresponding interrupt enable bit is set.

### 7.6.3.21 Data Channel Operation

The Host/Card relation on the data channel is defined to be a master-slave interface. The Host will periodically poll the Card to determine if it has data. The Card will only transmit data to the Host after one of these polls. The interrupts are particularly useful when the transaction has to be fragmented. The method of interrupt implementation is dependent on the Host manufacturer and is not defined in this document.

### 7.6.3.22 Extended Channel Operation

The Host/Card relation on the extended channel is defined in Section 9.14 of this document. This is a peer type interface. The Host and Card can transmit data over the extended channel at any time. The interrupt implementation is dependent on the Host manufacturer and is not defined in this document.

### 7.6.3.23 Priorities

Since the data and extended channel interrupts are logically OR'ed together to a single interrupt signal, a priority must be established. Since the data channel is defined to be the command interface, it SHALL have priority over the extended channel. Additionally, the data channel will have less traffic overall than the extended channel.

This priority can be easily implemented by having the Host first read the data channel status byte and then the extended channel status byte when an interrupt occurs to resolve the source.

### 7.6.4 M-CARD Initialization and Operation

For the M-CARD, the following is an example of operation of the CPU interface at initialization through opening the resource manager session. For the purpose of this example, it is assumed that the M-CARD and Host contains four buffers, one each for transmission and receiving for both channels, each buffer is 4,096 bytes. All flags are assumed to be initialized to zero and will remain unchanged unless specifically called out below. In the tables below, an X is defined to be “don't care” for the receiving device.

1. The Host brings RESET inactive. The Host SHALL bring its HR flag active within 1 second. It SHOULD be noted that it is expected for the Host to be fully operational prior to bringing RESET inactive.

**Table 7.6-27 - Buffer 1**

	Bit	7	6	5	4	3	2	1	0
Host	Query	X	HR = 1	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	0	0	0	0
M-CARD	Query	X	CR = 0	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	X	X	X	X	X	X	X	X
	Length LSB	X	X	X	X	X	X	X	X

2. The M-CARD brings its CR flag active within 5 seconds.

**Table 7.6-28 - Buffer 2**

	Bit	7	6	5	4	3	2	1	0
Host	Query	X	HR = 1	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	0	0	0	0
M-CARD	Query	X	CR = 1	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	X	X	X	X	X	X	X	X
	Length LSB	X	X	X	X	X	X	X	X

3. The M-CARD loads its open\_session\_request SPDU to open a session to the Host's resource manager



resource into its command channel output buffer.

4. The M-CARD sets its DA flag and clears its EC flag on the next Host query. Since there are 6 bytes in the open\_session\_request, it will set the length to 0x0006. The Host will read the 2<sup>nd</sup> and 3<sup>rd</sup> bytes of data from the M-CARD. The Host SHALL then clock in 6 data bytes from the M-CARD.

**Table 7.6–29 - Buffer 3**

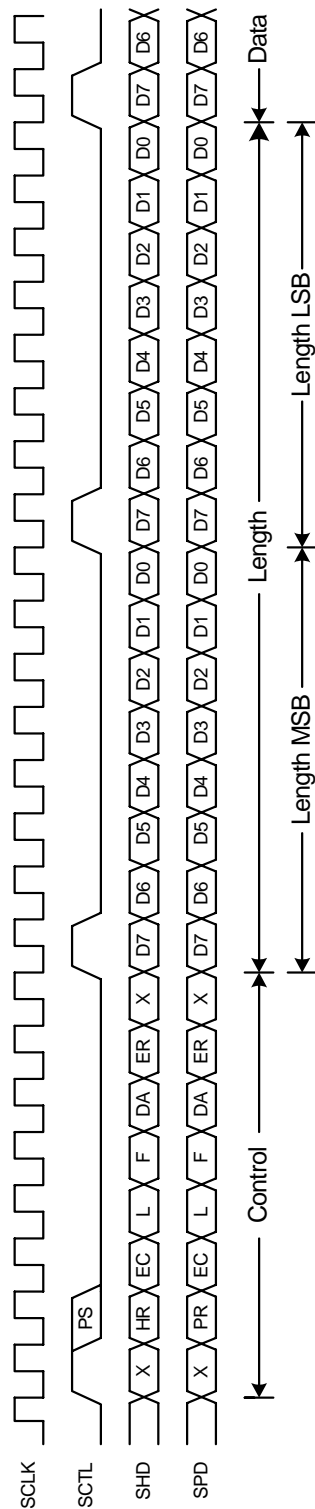
	Bit	7	6	5	4	3	2	1	0
Host	Query	X	HR = 1	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	0	0	0	0
M-CARD	Query	X	CR = 1	EC = 0	L = 1	F = 1	DA = 1	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	0	1	1	0

5. The Host reads the value in its command channel input buffer. The Host then responds by putting its open\_session\_response SPDU into its command channel output buffer. If the time to process this is significant, then the Host MAY clear its HR flag until it is ready to send/receive more data.
6. The Host then sets its DA flag and HR flag, and clears its EC flag on the next Host query. The 2<sup>nd</sup> and 3<sup>rd</sup> bytes contain the length of the data, which will be 0x0009 for the open\_session\_response SPDU. The Host will then clock out all data in its command channel output buffer. The M-CARD will clock this data into its command channel receive buffer.

**Table 7.6–30 - Buffer 4**

	Bit	7	6	5	4	3	2	1	0
Host	Query	X	HR = 1	EC = 0	L = 1	F = 1	DA = 1	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	1	0	0	1
M-CARD	Query	X	CR = 1	EC = X	L = X	F = X	DA = 0	ER = 0	X
	Length MSB	0	0	0	0	0	0	0	0
	Length LSB	0	0	0	0	0	0	0	0

7. The M-CARD then clears its ready flag, CR, while it processes the data in its buffer.
8. Now that the resource manager session is open, the M-CARD will now load the profile\_inq() APDU into its command channel output buffer and then bring the CR and DA flags active. From here, it will continue with the initialization procedure described in Sections 7.6.3.11.8 through 7.6.3.18.



**Figure 7.6-27 - M-Mode Serial Interface Protocol Diagram**

## 8 COPY PROTECTION

Copy protection SHALL be provided for content marked with a non-zero EMI delivered in MPEG transport streams flowing from the Card to the Host. Such protection, including scrambling of content from Card to Host and authenticated delivery of messages through the CPU interface for permitted use of 'content marked with a non-zero EMI, is defined in OpenCable CableCARD Copy Protection 2.0 Specification [CCCP2].

## 9 COMMAND CHANNEL OPERATION

The Command Channel is the control interface between the Card and the Host. S-Mode and M-Mode Command Channel (also known as Data Channel) operation employ identical Session Resource and Application layers.

S-Mode SHALL use the Link, and Transport layers as defined in Sections 5 and 7 of this document.

M-Mode, The Transport Layer as defined for S-Mode operation is not used, and the Link Layer Operation is implemented using the F and L bits in the Interface Query Byte. The Interface Query Byte functionality is further defined in Section 7.6.2.2.

### 9.1 Session Layer

The session layer provides a mechanism for establishing communications between applications on the Card and resources on the Host.

#### 9.1.1 S-Mode

The Session Layer provides the mechanism by which applications communicate with and make use of resources. The resource is a mechanism for encapsulating functionality at the Application Layer and is described in Section 9.3.

Resources vary in the number of simultaneous sessions they can support. Some resources support only one. If a second application tries to request a session to such a resource already in use then it will receive a 'resource busy' reply. Other resources can support more than one simultaneous session, in which case resource requests will be honored up to some limit defined by the resource. An example of the latter would be the display resource, which in some Host implementations may be able to support simultaneous displays in different windows.

#### 9.1.2 M-Mode

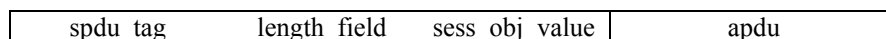
All sessions SHALL be opened by the Card applications. All resources defined by this specification, by definition, SHALL be resident in the Host.

#### 9.1.3 Resources with Multiple Sessions

Some resources MAY have multiple sessions so a method of identifying these is required. The session layer allows for this.

#### 9.1.4 SPDU Structure

The session layer uses a Session Protocol Data Unit (SPDU) structure to exchange data at session level either from the Host to the Card or from the Card to the Host. The general form of the SPDU structure is made of two parts, a mandatory session header, consisting of a tag value, a length field and the session object value, and the conditional body of variable length.



**Figure 9.1-1 - SPDU Structure**

The SPDU is made of two parts:

1. A mandatory session header made of a Tag value `spdu_tag`, a `length_field` coding the length of the session object value field and a session object value. Note that the length field does *not* include the length of any following APDUs.
2. A conditional body of variable length which contains a APDU. (see application layer). The presence of the body depends on the session header.

**Table 9.1–1 - SPDU Structure Syntax**

Syntax	No. of Bits	Mnemonic
SPDU() { spdu_tag length_field() for (i=0; i<length_value; i++) { session_object_value_byte } for (i=0; i<N; i++) { data_byte } }	8  8  8	uimsbf  uimsbf  uimsbf

**spdu\_tag** One of the values listed in Table 9.1–7.

**N** Variable, depending on the specific spdu\_tag.

A SPDU is transported in the data field of one TPDU.

Only one SPDU header is followed by a data field - the session\_number object - which is always followed by a SPDU body containing one APDU.

A SPDU is transported in the data field of one or several TPDU. See TPDU description of each physical module implementation for more information.

#### 9.1.4.1 Open Session Request (open\_session\_request)

This object is issued by the Card to the Host in order to request the opening of a session between the device and one resource provided by the Host. The resource\_identifier SHALL match in both class and type, a resource that the Host has in its list of available resources. If the version of the supplied resource identifier is zero, the Host will use the current version in its list. If the version number in the request is less than or equal to the current version number in the Host's list, the current version is used. If the requested version number is higher than the version in the Host's list, the Host will refuse the request with the appropriate return code.

**Table 9.1–2 - open\_session\_request() Syntax**

Syntax	No. of Bits	Mnemonic
open_session_request() { open_session_request_tag length_field() /* always equal to 0x04 */ resource_identifier() }	8	uimsbf

**open\_session\_request\_tag** 0x91

**resource\_identifier** See Table 9.3–2.

#### 9.1.4.2 Open Session Response (open\_session\_response)

This open\_session\_response SPDU is issued by the Host to the Card in order to allocate a session number or inform the Card that its request could not be met.

**Table 9.1–3 - open\_session\_response() Syntax**

Syntax	No. of Bits	Mnemonic
open_session_response() { open_session_response_tag length_field() /* always equal to 0x07 */ session_status resource_identifier() session_nb }	8  8  16	uimsbf  uimsbf  uimsbf

**open\_session\_response\_tag**      0x92

**session\_status**      Status of the open session request.

- 0x00    Session is opened
- 0xF0    Session not opened – resource non-existent or not supported
- 0xF1    Session not opened – resource exists but unavailable
- 0xF2    Session not opened – resource exists but version lower than requested
- 0xF3    Session not opened – resource busy
- 0x01-0xEF    Reserved
- 0xF4-0xFF    Reserved

**resource\_identifier**      See Table 9.3–2 for description. The Host returns the actual resource identifier of the resource requested with the current version number. If the session\_status response is 0xF0, ‘resource non-existent’, then the resource identifier field SHALL be identical to that supplied in the open\_request SPDU.

**session\_nb**      A 16-bit integer number allocated by the Host for the requested session. A value of 0x00 is reserved and SHALL NOT be used. The session\_nb SHALL be used for all subsequent exchanges of APDUs between the Card and the Host until the session is closed. When the session could not be opened (session\_status ≠ 0x00), this value has no meaning.

#### 9.1.4.3 Close Session Request (close\_session\_request)

The *close\_session\_request()* SPDU MAY be issued by the Host or the Card to close a session.

**Table 9.1–4 - close\_session\_request() Syntax**

Syntax	No. of Bits	Mnemonic
close_session_request() { close_session_request_tag length_field() /* always equal to 0x02 */ session_nb }	8  16	uimsbf  uimsbf

**close\_session\_request\_tag**      0x95

**session\_nb**      The 16-bit integer value assigned to the session.

#### 9.1.4.4 Close Session Response (close\_session\_response)

The Host or Card SHALL issue the close\_session\_response SPDU after receiving a close\_session\_request SPDU.

**Table 9.1–5 - close\_session\_response() Syntax**

Syntax	No. of Bits	Mnemonic
close_session_response() {		
close_session_response_tag	8	uimsbf
length_field() /* always equal to 0x03 */		
session_status	8	uimsbf
session_nb	16	uimsbf
}		

**close\_session\_response\_tag**      0x96

**session\_status**                      Status of the close session request.

   0x00    Session is closed as required

   0xF0    session\_nb in the request is not allocated

   0x01-0xEF    Reserved

   0xF1-0xFF    Reserved

**session\_nb**                              The 16-bit integer value assigned to the session.

#### 9.1.4.5 Session Number (session\_number)

The session\_number SPDU SHALL always precede a body of the SPDU containing an APDU.

**Table 9.1–6 - session\_number() Syntax**

Syntax	No. of Bits	Mnemonic
session_number() {		
session_number_tag	8	uimsbf
length_field() /* always equal to 0x02 */		
session_nb	16	uimsbf
}		

**session\_number\_tag**      0x90

**session\_nb**                              The 16-bit integer value assigned to the session.

#### 9.1.4.6 Summary of Session Tags (spdu\_tag)

**Table 9.1–7 - Summary of SPDU Tags**

spdu_tag	tag value	Direction Host ↔ Card
open_session_request	0x91	←
open_session_response	0x92	→
create_session	0x93	→
create_session_response	0x94	←
close_session_request	0x95	↔
close_session_response	0x96	↔
session_number	0x90	↔

All other values are reserved.

## 9.2 Application Layer

The application layer implements a set of protocols based upon the concept of a resource. A resource defines a unit of functionality which is available to applications running on a Card. Each resource supports a set of objects and a protocol for interchanging them to use the resource. Communication with a resource is by means of a session created to that particular resource.

Resources are provided by the Host. Resources are used by an application creating a session to a resource. By an initialization process carried out by the Resource Manager, the Host identifies all available resources and can complete the session. Once this session has been created the application can then use the resource by an exchange of objects according to the defined protocol.

By definition, applications reside on the Card and resources reside on the Host.

### 9.2.1 Resource Identifier Structure

A resource identifier consists of 4 octets. The two most significant bits of the first octet indicate whether the resource is public or private. Values of 0, 1, and 2 indicate a public resource. A value of 3 indicates a private resource.

Public resource is divided into three components: resource class, resource type, and resource version. Resource class defines a set of objects and a protocol for using them. Resource type defines distinct resource units within a class. All resource types within a class use the same objects and protocol, but offer different services or are different instances of the same service. Resource version allows the Host to identify the latest version (highest version number) of a resource where more than one of the same class and type are present. This allows updated or enhanced resource to be supplied on a Card to supersede existing resources in the Host. Resources with a higher version number SHALL be backward compatible with previous versions, so that applications requesting a previous version will have a resource with expected behavior.

Public resource classes have values allocated in the range 1 to 49150, treating the `resource_id_type` field as the most significant part of `resource_class`. Value 0 is reserved. The maximum (all-ones) value of all fields is reserved. Private resources are identified by a private resource definer, that is, the organization that defines a private resource. Each private resource definer can define the structure and content of the `private_resource_identity` field in any way except that the maximum (all-ones) value is reserved.

**Table 9.2–1 - Public Resource Identifier**

Bit																																	
3							2	2							1	1										8	7						0
1							4	3							6	5																	
type		resource class														resource type										resource version							

**Table 9.2–2 - Private Resource Identifier**

Bit																																				
31						24	23									1	1											8	7							0
3		private resource definer										private resource identity																								



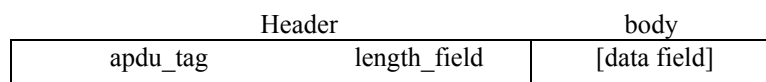
**Table 9.2–3 - resource\_identifier() Syntax**

Syntax	No. of Bits	Mnemonic
resource_identifier() { resource_id_type if (resource_id_type != 0x3) { resource_class resource_type resource_version } else { private_resource_definer private_resource_identity } }	2  14 10 6  10 20	uimsbf  uimsbf uimsbf uimsbf  uimsbf uimsbf

These are the two types of application layer entity that can reside on a Card. Applications make use of resources to perform tasks for the user of the Host. Resource providers provide resources additional to those available directly in the Host, or a newer version of a resource replacing one previously provided by the Host. To avoid deadlock problems and complexities of initialization resource providers SHALL NOT be dependent on the presence of any other resources, except the Resource Manager, to provide the resources they offer.

### 9.3 APDUs

All protocols in the Application Layer use a common Application Protocol Data Unit (APDU) structure to send application data between Host and the Card. The APDU is made up of two parts, a mandatory header, consisting of a the apdu\_tag value and a length field coding the length of the following data field and the conditional body of variable length.

**Figure 9.3-1 - APDU Structure****Table 9.3–1 - APDU Structure Syntax**

Syntax	No. of Bits	Mnemonic
APDU() { apdu_tag length_field() for (i=0; i<length_value; i++) { data_byte } }	24  8	uimsbf  uimsbf

Chaining of APDUs SHALL NOT be supported.

Table 9.3–2 is a summary of all of the supported Resource Identifiers.

**Table 9.3–2 - Resource Identifier Values**

Resource	Class	Type	Version	Resource identifier
Resource Manager	1	1	1	0x00010041
Application Information	2	2	1	0x00020081
Conditional Access Support	3	1	2	0x00030042
Conditional Access Support*	3	2	1	0x00030081
Host Control	32	1	3	0x00200043
Host Control*	32	2	1	0x00200081
System Time	36	1	1	0x00240041
System Time	36	1	2	0x00240042
MMI	64	2	1	0x00400081
MMI*	64	3	1	0x004000C1
Low Speed Communication <sup>2</sup>	96	321	3	0x00605043
Low Speed Communication <sup>2</sup>	96	513	3	0x00608043
Homing <sup>1</sup>	17	1	2	0x00110042
Copy Protection	176	3	1	0x00B000C1
Copy Protection*	176	4	1	0x00B00101
Specific Application Support	144	1	2	0x00900042
Generic Feature Control	42	1	1	0x002A0041
Extended Channel	160	1	1	0x00A00041
Extended Channel	160	1	2	0x00A00042
Extended Channel	160	1	3	0x00A00043
Extended Channel	160	1	4	0x00A00044
Generic IPPV Support <sup>3</sup>	128	2	1	0x00800081
Generic Diagnostic Support	260	1	1	0x01040041
Generic Diagnostic Support*	260	2	1	0x01040081
System Control	43	1	1	0x002B0041
CARD RES*	38	3	1	0x002600C1
Reserved**	177	1	1	0x00B10041
Notes: *For the Card operating in M-Mode. **Reserved. 1. The Homing resource is defined in 9.18.4. 2. The Resource identifier delivered by a Host SHALL be either 0x00605043 for a Host device with Cable Return Channel, or 0x00608043 for a Host with a Host Modem (e.g., DOCSIS). If no Low Speed Communication Resource Identifier reported by the Host then the Host device is assumed to be a FDC only. The Card MAY utilize the presence of this resource identifier as a means to identify what type of Cable Return Channel is supported by the Host. 3. If a device manufacturer opts to implement an optional resource on a device, then the resource SHALL support the baseline resource ID.				

The coding of the apdu\_tag follows the ASN.1 rules. Each apdu\_tag is coded on three bytes. Among the 24 bits of each apdu\_tag, 10 are fixed by the ASN.1 rules as described in Figure 9.3-2. Only primitive tags are used.

Byte 1				Byte 2				Byte 3			
b24		b17	b16		b9			b8			b1
1	0	0	1	1	1	1	1	1	x	x	x

**Figure 9.3-2 - Primitive Tag Coding**

Table 9.3–3 is a summary of all of the supported APDUs.

**Table 9.3–3 - Application Object Tag Values**

<b>apdu_tag</b>	<b>Tag value</b>	<b>Resource</b>	<b>Direction Host ↔ Card</b>	
profile_inq	0x9F8010	Resource Manager	←	
profile_reply	0x9F8011	Resource Manager	→	
profile_changed	0x9F8012	Resource Manager	→	
application_info_req	0x9F8020	Application Info	→	
application_info_cnf	0x9F8021	Application Info	←	
server_query	0x9F8022	Application Info	→	
server_reply	0x9F8023	Application Info	←	
ca_info_inq	0x9F8030	CA Support	→	
ca_info	0x9F8031	CA Support	←	
ca_pmt	0x9F8032	CA Support	→	
ca_pmt_reply	0x9F8033	CA Support	←	
ca_update	0x9F8034	CA Support	←	
oob_tx_tune_req	0x9F8404	Host Control	←	
oob_tx_tune_cnf	0x9F8405	Host Control	→	
oob_rx_tune_req	0x9F8406	Host Control	←	
oob_rx_tune_cnf	0x9F8407	Host Control	→	
inband_tune_req	0x9F8408	Host Control	←	
inband_tune_cnf	0x9F8409	Host Control	→	
system_time_inq	0x9F8442	System Time	←	
system_time	0x9F8443	System Time	→	
open_mmi_req	0x9F8820	MMI	←	
open_mmi_cnf	0x9F8821	MMI	→	
close_mmi_req	0x9F8822	MMI	←	
close_mmi_cnf	0x9F8823	MMI	→	
comms_cmd*	0x9F8C00	Low speed comms.	←	
connection_descriptor*	0x9F8C01	Low speed comms.	←	
comms_reply*	0x9F8C02	Low speed comms.	→	
comms_send_last*	0x9F8C03	Low speed comms.	←	
comms_send_more*	0x9F8C04	Low speed comms.	←	
comms_rcv_last*	0x9F8C05	Low speed comms.	→	
comms_rcv_more*	0x9F8C06	Low speed comms.	→	
			<b>Host modem</b>	<b>Card Modem</b>
new_flow_req	0x9F8E00	Extended Channel Support	↔	→
new_flow_cnf	0x9F8E01	Extended Channel Support	↔	←
delete_flow_req	0x9F8E02	Extended Channel Support	↔	→
delete_flow_cnf	0x9F8E03	Extended Channel Support	↔	←
lost_flow_ind	0x9F8E04	Extended Channel Support	↔	←
lost_flow_cnf	0x9F8E05	Extended Channel Support	↔	→
inquire_DSG_mode	0x9F8E06	Extended Channel Support	→	N/A

apdu_tag	Tag value	Resource	Direction Host ↔ Card	
			←	→
set_DSG_mode	0x9F8E07	Extended Channel Support	←	N/A
DSG_error	0x9F8E08	Extended Channel Support	←	N/A
dsg_message	0x9F8E09	Extended Channel Support	→	N/A
configure_advanced_DSG	0x9F8E0A	Extended Channel Support	←	N/A
send_DCD-info	0x9F8E0B	Extended Channel Support	→	N/A
Reserved	0x9F8F00 – 0x9F8F07	Generic IPPV Support		
feature_list_req	0x9F9802	Generic Feature Control	↔	
feature_list	0x9F9803	Generic Feature Control	↔	
feature_list_cnf	0x9F9804	Generic Feature Control	↔	
feature_list_changed	0x9F9805	Generic Feature Control	↔	
feature_parameters_req	0x9F9806	Generic Feature Control	←	
feature_parameters	0x9F9807	Generic Feature Control	↔	
features_parameters_cnf	0x9F9808	Generic Feature Control	↔	
open_homing	0x9F9990	Homing	→	
homing_cancelled	0x9F9991	Homing	→	
open_homing_reply	0x9F9992	Homing	←	
homing_active	0x9F9993	Homing	→	
homing_complete	0x9F9994	Homing	←	
firmware_upgrade	0x9F9995	Homing	←	
firmware_upgrade_reply	0x9F9996	Homing	→	
firmware_upgrade_complete	0x9F9997	Homing	←	
SAS_connect_rqst	0x9F9A00	Specific Application Support	→	
SAS_connect_cnf	0x9F9A01	Specific Application Support	←	
SAS_data_rqst	0x9F9A02	Specific Application Support	↔	
SAS_data_av	0x9F9A03	Specific Application Support	↔	
SAS_data_cnf	0x9F9A04	Specific Application Support	↔	
SAS_data_query	0x9F9A05	Specific Application Support	↔	
SAS_data_reply	0x9F9A06	Specific Application Support	↔	
SAS_async_msg()	0x9F9A07	Specific Application Support	↔	
stream_profile()	0x9FA010	CARD RES	←	
stream_profile_cnf()	0x9FA011	CARD RES	→	
program_profile()	0x9FA012	CARD RES	←	
program_profile_cnf()	0x9FA013	CARD RES	→	
es_profile()	0x9FA014	CARD RES	←	
es_profile_cnf()	0x9FA015	CARD RES	→	
request_pids()	0x9FA016	CARD RES	←	

<b>apdu_tag</b>	<b>Tag value</b>	<b>Resource</b>	<b>Direction Host ↔ Card</b>
request_pids_cnf()	0x9FA017	CARD RES	→
asd_registration_req **	0x9FA200	Authorized Service Domain	→
asd_challenge**	0x9FA201	Authorized Service Domain	←
asd_challenge_rsp**	0x9FA202	Authorized Service Domain	→
asd_registration_grant**	0x9FA203	Authorized Service Domain	←
asd_dvr_record_req**	0x9FA204	Authorized Service Domain	→
asd_dvr_record_reply**	0x9FA205	Authorized Service Domain	←
asd_dvr_playback_req**	0x9FA206	Authorized Service Domain	→
asd_dvr_playback_reply**	0x9FA207	Authorized Service Domain	←
asd_dvr_release_req**	0x9FA208	Authorized Service Domain	→
asd_dvr_release_reply**	0x9FA209	Authorized Service Domain	←
asd_server_playback_req**	0x9FA20A	Authorized Service Domain	→
asd_server_playback_reply**	0x9FA20B	Authorized Service Domain	←
asd_client_playback_req**	0x9FA20C	Authorized Service Domain	→
asd_client_playback_reply**	0x9FA20D	Authorized Service Domain	←
host_info_request	0x9F9C00	System Control	←
host_info_response	0x9F9C01	System Control	→
code_version_table	0x9F9C02	System Control	←
code_version_table_reply	0x9F9C03	System Control	→
host_download_control	0x9F9C04	System Control	→
host_download_command	0x9F9C05	System Control	←
diagnostic_req	0x9FDF00	Generic Diagnostic Support	←
diagnostic_cnf	0x9FDF01	Generic Diagnostic Support	→

\* Messages defined in EIA 679-B Part B [NRSSB].

\*\*Reserved.

For Transportation of APDU within SPDU, Only a single APDU SHALL be supported in the body of an SPDU.

### 9.3.1 Interface Resource Loading

**Table 9.3–4 - Host-Card Interface Resource Loading**

Item	Name	Maximum sessions at one time	Closes	Resource Location
1	Transport Connection ID	1	No	Host creates TC_ID
2	Sessions total (sum of items 3-16)	128	N/A	N/A
3	Resource Manager	32	No	Host
4	MMI	1	No	Host
5	Application Info	1	No	Host
6	Low Speed Communication	1	Yes	Host
7	Conditional Access Support	1	No	Host
8	Copy Protection	1	No	Host
9	Host Control	1	No	Host
10	Extended Channel Support	2	No	Host
11	Generic IPPV Support	1	Yes	Host
12	Specific Application Support	32	Yes	Host
13	Generic Feature Control	1	No	Host
14	Homing	1	Yes	Host
15	Generic Diagnostic Support	1	No	Host
16	System Time	1	Yes	Host
17	System Control	1	No	Host
18	CARD RES (Card Resource)	1	No	Host
19	Authorized Service Domain	1	No	Host

#### NOTES:

A maximum of one Generic Diagnostic Support resource may be open at a time.

After buffer negotiation, the Host will create a transport connection. The Card will ignore the t\_c\_id value in the link layer when there is no transport connection established.

In S-Mode, only one program may be descrambled at a time, hence only one conditional access support session SHALL be opened. The conditional access session does not close.

In M-Mode, one or more programs may be descrambled at a time, only one conditional access support session SHALL be opened. The conditional access session does not close.

Only one copy protection session SHALL be open at a time. The copy protection session does not close.

Two extended channel sessions MAY be open at a time, but only when the Host has modem capability (either phone, DOCSIS, or other); otherwise only one extended channel session MAY be open. The extended channel session does not close.

Only one host control session SHALL be open at a time. The host control session does not close.

A maximum of one Generic IPPV session MAY be open at a time.

A maximum of one homing session MAY be open at a time.

A maximum of one system time session MAY be open at a time.

The Card SHALL limit the resource to the values defined in Table 9.3–2. The Host SHALL at a minimum support the number of resources defined in Table 9.3–2.

## 9.4 Resource Manager

The Resource Manager is a resource provided by the Host. There is only one type in the class. A maximum of 32 sessions SHALL be supported. It controls the acquisition and provision of resources to all applications. A discovery mechanism exists for the Card to determine which resources as well as the versions are supported on the Host.

It SHALL be mandatory that the first session opened be to the Resource Manager. After the session has been opened, the Card SHALL transmit a *profile\_inq()* APDU to the Host. The Host SHALL respond with the *profile\_reply()* APDU listing all available resources. The Card SHALL be able to issue a *profile\_inq()* APDU at any time to which the Host SHALL respond.

In the rare event that a resource is modified on the Host, the Host SHALL issue a *profile\_changed()* APDU to the Card. The Card SHALL then issue a *profile\_inq()* APDU to the Host. The Host SHALL then issue the *profile\_reply()* APDU. It SHALL be the Card's responsibility to determine if a resource has been modified such that the session should be closed and reopened.

**Table 9.4–1 - Resource Manager Resource Identifier**

Resource	Mode	Class	Type	Version	Identifier (hex)
Resource Manager	S-Mode/M-Mode	1	1	1	0x00010041

The Resource Manager includes three APDUs as described in the following table:

**Table 9.4–2 - Resource Manager APDU List**

APDU Name	Tag Value	Resource	Direction Host ↔ Card
profile_inq()	0x9F8010	Resource Manager	←
profile_reply()	0x9F8011	Resource Manager	→
profile_changed()	0x9F8012	Resource Manager	→

### 9.4.1 profile\_inq()

The *profile\_inq()* APDU is issued by the Card to request the Host to transmit its available resources in the *profile\_reply()* APDU.

**Table 9.4–3 - profile\_inq() APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>profile_inq() {   profile_inq_tag   length_field() /* always = 0x00 */ }</pre>	24	uimbsf

**profile\_inq\_tag**            0x9F8010

### 9.4.2 profile\_reply()

The *profile\_reply()* APDU is issued by the Host in response to the *profile\_inq()* APDU from the Card.

**Table 9.4–4 - profile\_reply() APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>profile_reply() {   profile_reply_tag   length_field()   for (i=0; i&lt;N; i++) {     resource_identifier()   } }</pre>	24	uimsbf

**profile\_reply\_tag**      0x9F8011

**N**      length divided by 4

### 9.4.3 profile\_changed()

The *profile\_changed()* APDU is transmitted to the Host if the value of any resource identifier has changed.

**Table 9.4–5 - profile\_changed() APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>profile_changed() {   profile_changed_tag   length_field() /* always = 0x00 */ }</pre>	24	uimsbf

**profile\_changed\_tag**      0x9F8012

## 9.5 Application Information

The Application Information resource resides in the Host. The Card SHALL only open one session to it after it has completed the profile inquiry operation with the Resource Manager resource.

The Application Information resource provides:

- Support for the Host to expose its display characteristics to the Card
- Support for the Card to expose its applications to the Host
- Support for the Card to deliver HTML pages to the Host

The Application Information resource has been changed to type 2 to reflect the changes listed in this section as compared to [NRSSB]. (The Card is not required to support type 1.) During initialization, the Card opens a session to the Application Information resource on the Host. This session SHALL remain open during normal operation.

**Table 9.5–1 - Application Information Resource Identifier**

Resource	Mode	Class	Type	Version	Identifier (hex)
Application Info	S-Mode/M-Mode	2	2	1	0x00020081

The Application Information resource includes four APDUs as described in Table 9.5–2.



**Table 9.5–2 - Application Information APDU List**

APDU Name	Tag Value	Resource	Direction Host ↔ Card
application_info_req()	0x9F8020	Application Info	→
application_info_cnf()	0x9F8021	Application Info	←
server_query()	0x9F8022	Application Info	→
server_reply()	0x9F8023	Application Info	←

The method for initiating an application is beyond the scope of this document.

### 9.5.1 application\_info\_req()

After the Card has opened the Application Information resource, the Host SHALL send an *application\_info\_req()* APDU to the Card. This SHALL include the display capabilities of the Host. The Card SHALL reply with an *application\_info\_cnf()* APDU to describe its support applications. The Host MAY send this APDU anytime later to which the Card SHALL reply.

**Table 9.5–3 - application\_info\_req() APDU Syntax**

Syntax	No. of Bits	Mnemonic
application_info_req() { application_info_req_tag length_field() display_rows display_columns vertical_scrolling horizontal_scrolling display_type_support data_entry_support HTML_support if (HTML_support == 1) { link_support form_support table_support list_support image_support } }	24  16 16 8 8 8 8 8 8  8 8 8 8 8	uimbsbf  uimbsbf uimbsbf uimbsbf uimbsbf uimbsbf uimbsbf uimbsbf  uimbsbf uimbsbf uimbsbf uimbsbf uimbsbf

**application\_info\_req\_tag**      0x9F8020

**display\_rows**      Defines the number of rows the Host device can support. If the Host supports more than 255, set this value equal to 0xFF.

**Display\_columns**      Defines the number of columns the Host device can support. If the Host supports more than 255, set this value equal to 0xFF.

**Vertical\_scrolling**      Defines if the Host supports vertical scrolling. Default value is 0.  
                                  0x00    Vertical scrolling not supported  
                                  0x01    Vertical scrolling supported  
                                  0x02-0xFF    Reserved

**horizontal\_scrolling**      Defines if the Host supports horizontal scrolling. Default value is 0.  
                                  0x00    Horizontal scrolling not supported  
                                  0x01    Horizontal scrolling supported

	0x02-0xFF	Reserved
<b>display_type_support</b>	Defines the window support capability of the Host.	
	0x00	Full screen. The Host supports full screen windows for MMI screens.
	0x01	Overlay. The Host supports Overlay Windows for MMI screens.
	0x02-0x6F	Multiple windows. Indicates that the Host supports multiple simultaneous MMI windows. The value equals the maximum number of simultaneous open windows the Host can support.
	0x70-0xFF	Reserved
	This field defines the type of windowing the Host supports, and if the Host supports multiple windows, the number of simultaneous windows the Host display application can manage.	
	If the Host supports the Full Screen Window Type and if that Window Type is requested by the Card via the <i>open_mmi_req()</i> APDU, the Host SHALL present the MMI dialog in an opaque window.	
	If the Host supports the Overlay Window Type and if that Window Type is requested by the Card via the <i>open_mmi_req()</i> APDU, the Host SHALL present the MMI dialog in an opaque window which is large enough to contain the MMI message but may not fill the entire viewable area.	
	If the Host supports the Multiple Windows Type and if that Window Type is requested by the Card via the <i>open_mmi_req()</i> APDU, the Host SHALL present each subsequent MMI dialog, up to the maximum number conveyed, in an Overlay Window. The Host SHALL support a method for navigating between the different windows.	
<b>Data_entry_support</b>	Defines the preferred data entry capability of the Host.	
	0x00	None
	0x01	Last/Next
	0x02	Numeric Pad
	0x03	Alpha keyboard with mouse
	0x04-0xFF	Reserved
<b>HTML_support</b>	Defines the HTML support capability of the Host. All Hosts SHALL support at a minimum the Baseline HTML profile. The Baseline HTML profile is defined in Annex A.	
	0x00	Baseline Profile
	0x01	Custom Profile
	0x02	HTML 3.2
	0x03	XHTML 1.0
	0x04-0xFF	Reserved
<b>link_support</b>	Defines whether the Host can support single or multiple links.	
	0x00	One link
	0x01	Multiple links
	0x02-0xFF	Reserved
<b>form_support</b>	Defines the Form support capability of the Host.	
	0x00	None
	0x01	HTML 3.2 w/o POST method
	0x02	HTML 3.2
	0x03-0xFF	Reserved
<b>table_support</b>	Defines the Table support capability of the Host.	
	0x00	None
	0x01	HTML 3.2
	0x02-0xFF	Reserved

**list\_support** Defines the List support capability of the Host

0x00 None  
 0x01 HTML 3.2 w/o Descriptive Lists  
 0x02 HTML 3.2  
 0x03-0xFF Reserved

**image\_support** Defines the Image capability of the Host

0x00 None  
 0x01 HTML 3.2 – PNG Picture under RGB w/o resizing  
 0x02 HTML 3.2  
 0x03-0xFF Reserved

## 9.5.2 application\_info\_cnf()

The Card SHALL transmit the *application\_info\_cnf()* APDU after receiving an *application\_info\_req()* from the Host.

**Table 9.5–4 - application\_info\_cnf() APDU Syntax**

Syntax	No. of Bits	Mnemonic
application_info_cnf() {		
application_info_cnf_tag	24	uimsbf
length_field()		
CableCARD_manufacturer_id	16	uimsbf
CableCARD_version_number	16	uimsbf
number_of_applications	8	uimsbf
for (i=0; i<number_of_applications; i++) {		
application_type	8	uimsbf
application_version_number	16	uimsbf
application_name_length	8	uimsbf
for (j=0; j<application_name_length; j++) {		
application_name_byte	8	uimsbf
}		
application_url_length	8	uimsbf
for (j=0; j<application_url_length; j++) {		
application_url_byte	8	uimsbf
}		
}		
}		

**application\_info\_cnf\_tag** 0x9F8021

**CableCARD\_manufacturer\_id** The first byte specifies the Card manufacturer while the second byte is defined by the Card manufacturer to privately identify product generation and derivatives.

0x00XX Motorola  
 0x01XX Scientific-Atlanta  
 0x02XX SCM Microsystems  
 0x0300-0xFFFF Reserved for future manufacturers

**manufacturer\_version\_number** Privately defined by the Card manufacturer.

**Number\_of\_applications** Number of applications in the following for loop.

**Application\_type** Type of application.

0x00 Conditional access  
 0x01 CableCARD binding information application  
 0x02 IP service  
 0x03 Network interface [SCTE55-2]  
 0x04 Network interface [SCTE55-1]

	0x05 Copy protection application
	0x06 Diagnostic
	0x07 Undesignated
	0v08 Network Interface (DSG)
	0x09-0xFF Reserved for future applications
<b>application_version_number</b>	Defined by the Card application supplier.  The Card SHALL upgrade the <code>application_version_number</code> each time the Card application software is modified according to the Card Firmware Upgrade Host Interface. See Section 9.18 of this document.
<b>Application_name_length</b>	Length of the application name in the following for loop. The maximum value is 32. This value SHALL be equal to 0 for applications that do not have an MMI interface.
<b>Application_name_byte</b>	The commercial name of the application specified as a text string in ASCII format. The Host SHALL replace the default generic identifier of the Card's application with the application name. The application name, when selected by the user, triggers a Host initialized MMI dialog.  The Host SHALL be capable of displaying at least eight different Card application name strings in its top menu. The application name length SHALL be limited to 32 characters.
<b>Application_url_length</b>	Length of the application url in the following for loop.
<b>Application_url_byte</b>	Defines the URL of the Card application's top-level HTML page in the Card memory. The application URL MAY or MAY NOT be displayed in the Host top menu. The Host SHALL use the application URL in a <i><code>server_query()</code></i> APDU to initialize an MMI dialog with the Card application, when an object identified by either the application name or the application URL is selected in the Host menu.

### 9.5.3 `server_query()`

The Host SHALL send a *`server_query()`* APDU to the Card to request the information in the Card file server system pointed by a specific URL. The URL defines the location of the data that the Host is requesting. Upon receipt of the URL, the Card locates the requested data and provides it back to the Host in the *`server_reply()`* APDU. The Host SHALL process and display the data returned in the *`server_reply()`* APDU in a timely manner.

**Table 9.5–5 - `server_query()` APDU Syntax**

Syntax	No. of Bits	Mnemonic
<code>server_query() {</code>		
<code>server_query_tag</code>	24	<code>uimsbf</code>
<code>length_field()</code>		
<code>transaction_number</code>	8	<code>uimsbf</code>
<code>header_length</code>	16	<code>uimsbf</code>
for (i=0; i<header_length; i++) {		
<code>header_byte</code>	8	<code>uimsbf</code>
}		
<code>url_length</code>	16	<code>uimsbf</code>
for (i=0; i<url_length; i++) {		
<code>url_byte</code>	8	<code>uimsbf</code>
}		
<code>}</code>		

**server\_query\_tag**                      0x9F8022

<b>transaction_number</b>	A number supplied by the Host issued from an 8-bit cyclic counter that identifies each <i>server_query()</i> APDU and allows the Host to route the server_reply to the correct MMI dialog
<b>header_length</b>	The number of header bytes in the following for loop.
<b>Header_byte</b>	Each header_byte is an octet of an optional parameter that uses the same format as the HTTP/1.1 request header to pass additional parameters related to the request, like browser version, accepted mime types, etc. A Host not supporting headers SHALL set header_length to 0x00. The Card MAY also ignore this parameter.
<b>url_length</b>	The number of URL bytes in the following for loop.
<b>url_byte</b>	Each url_byte is an octet of a parameter that defines a protocol, domain, and location for the transfer of data. For the purposes of an application running on a Card, the URL SHALL allow the transfer of a file of data from the Card to the Host.

The access indicator is “CableCARD”.

The second part of the URL is the Host. The convention for “current server” (i.e., the server that generated the current page) can be used and is indicated by an empty Host.

The third part of the URL is the file location. This is indicated by a hierarchical directory/file path.

For example, in order to request the file menu.html from the directory/apps/user/program\_guide on the Card, the properly constructed URL would be:

CableCARD///apps/user/program\_guide/menu.html

If, after receiving a server\_reply from the Card, the Host has data that it wants to send to the Card, the Host can do so through a server\_query. In this case, the last part of the URL contains a list of name-value pairs separated by “&”. This list is preceded by “?”. A properly constructed URL would be:

CableCARD///path/file?name1=value1&name2=value2&...

Such a URL sent to an application on the Card as a response to a server\_reply would cause the name-value pairs to be processed by the application. Data entered and selected by the Host MAY be sent to the Card through the use of these name-value pairs as part of the URL in a *server\_query()* APDU.

#### 9.5.4 server\_reply()

The *server\_reply()* is issued by the Card in response to a *server\_query()* from the Host.

**Table 9.5–6 - server\_reply() APDU Syntax**

Syntax	No. of Bits	Mnemonic
server_reply() { server_reply_tag length_field() transaction_number file_status header_length for (i=0; i<header_length; i++) { header_byte } file_length for (i=0; i<url_length; i++) { file_byte } }	24   8 8 16  8  16  8	uimsbf   uimsbf uimsbf uimsbf  uimsbf  uimsbf  uimsbf

**server\_reply\_tag**

0x9F8023

**transaction\_number**

A number supplied by the Host issued from an 8-bit cyclic counter that identifies each *server\_query()* APDU and allows the Host to route the server\_reply to the correct MMI dialog.

**File\_status**

Notifies the Host of the status of the requested file.

0x00 OK  
 0x01 URL not found  
 0x02 URL access not granted  
 0x03-0xFF Reserved

**header\_length**

The number of header bytes in the following for loop.

**Header\_byte**

Each header\_byte is an octet of an optional parameter that uses the same format as the HTTP/1.1 request header to pass additional parameters related to the request, like browser version, accepted mime types, etc. A Host not supporting headers SHALL set header\_length to 0x00. The Card MAY also ignore this parameter.

**File\_length**

The number of bytes in the following for loop.

**File\_byte**

The requested URL file. A server reply object with file\_length equals to 0 will be interpreted as a null file.

## 9.6 Low Speed Communication

The Low Speed Communication resource is used to support the identification of the Forward Data Channel (FDC), the Reverse Data Channel (RDC), and any type of Host modem implementations. The Low Speed Communication resource is not a means for passing upstream/downstream OOB data to/from the Card via the CHI. All downstream OOB data SHALL be passed directly to/from the Card via the OOB Interface.

**Table 9.6–1 - A Low Speed Communication Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Low_Speed_Communication (Cable Return)	S-Mode/M-Mode	96	321	3	0x00605043
Low_Speed_Communication (DOCSIS Modem)	S-Mode/M-Mode	96	513	3	0x00608043

The Low\_Speed\_Communication Identifier SHALL be either 0x00605043 for a Host device with Cable Return Channel, (e.g., SCTE 55-1 or SCTE 55-2) or 0x00608043 for a Host with a Host Modem (e.g., DOCSIS).

A Host that has a DOCSIS Cable Modem but does not have an SCTE 55 transmitter is not supported and SHALL NOT report a Low Speed Communication identifier of either 0x00605043 or 0x00608043.

The following table summarizes this operation:

**Table 9.6–2 - Low-Speed Communication Resource ID Reporting Matrix**

SCTE 55 Receiver (FDC)	SCTE 55 Transmitter (RDC)	DOCSIS Cable Modem	Low Speed Communication Resource ID
Yes	No	No	None
Yes	No <sup>1</sup>	Yes	None
Yes	Yes	No	0x00605043
Yes	Yes	Yes	0x00608043

## 9.7 CA Support

This resource provides a set of objects to support Conditional Access applications. The Card will open a single session after the Application Information session initialization has completed. Like the resource manager, it is provided only by the Host and SHALL limit the session to one. The Host sends a CA Info inquiry object to the application, which responds by returning a CA Info object with the appropriate information. This session is then kept open for periodic operation of the protocol associated with the CA PMT and CA PMT Reply objects.

In the case of multiple transport streams, there is a local transport stream ID (LTSID) included in the APDU. The Host will then send a *ca\_info\_inquiry()* APDU to the Card, which will respond by returning a *ca\_info()* APDU with the appropriate information. The session SHALL be kept open. The Resource Type has been changed to indicate that the resource has been modified.

The **Conditional Access Support** resource SHALL be implemented in its entirety, with the following additions:

The Card SHALL inform the Host of changes in CA states by sending the *ca\_update()* APDU defined below. A new version of the CA resource on the Host SHALL process the *ca\_update()* APDU.

This APDU is modified from the PCMCIA Card to support the local transport stream ID (LTSID).

In M-Mode, each transport stream SHALL be controlled by the *ca\_pmt()* APDU with the associated transport stream id (LTSID). It is the Host's responsibility to manage the PID/program resources in the Card. The Host receives this information utilizing the *stream\_profile()* APDU, how many programs via *program\_profile()* APDU, and how many elementary streams via *es\_profile()* APDU as defined in Section 9.12. If the Host performs any filtering of elementary streams, it SHALL utilize the, *request\_pids()* and *request\_pids\_cnf()* APDUs as defined in Section 9.12.

**Table 9.7–1 - C A Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
CA Support	S-Mode	3	1	2	0x00030042
CA Support	M-Mode	3	2	1	0x00030081

The CA support resource consists of 5 APDUs for the S-Mode and the M-Mode.

<sup>1</sup> This variation is not permitted.

**Table 9.7–2 - CA Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
ca_info_inquiry()	0x9F8030	CA Support	→
ca_info()	0x9F8031	CA Support	←
ca_pmt()	0x9F8032	CA Support	→
ca_pmt_reply()	0x9F8033	CA Support	←
ca_update()	0x9F8034	CA Support	←

**9.7.1 ca\_info\_inquiry**

The *ca\_info\_inquiry()* APDU is transmitted by the Host to the Card after the CA session is opened.

**Table 9.7–3 - ca\_info\_inquiry() APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>ca_info_inquiry() {     ca_info_inquiry_tag     length_field() /* always = 0 */ }</pre>	24	uimsbf

**ca\_info\_inquiry\_tag**      0x9F8030

**9.7.2 ca\_info**

After the Card receives a *ca\_info\_inquiry()* APDU, it SHALL respond with the *ca\_info()* APDU with the appropriate CA\_system\_id.

**Table 9.7–4 - ca\_info() APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>ca_info() {     ca_info_tag     length_field()     for (i=0; i&lt;N ;i++) {         CA_system_id     } }</pre>	24	uimsbf
	16	uimsbf

**ca\_info\_tag**      0x9F8031

**CA\_system\_id**      CA system Ids supported by the Card.

**9.7.3 ca\_pmt**

The *ca\_pmt()* APDU consists of information extracted from the Program Map Table (PMT) in the PSI information by the Host and sent to the Card. This information contains all the control information to allow the Card to filter the ECMs itself and to make itself the correct assignment of an ECM stream with a scrambled component.

The *ca\_pmt()* APDU contains all the CA\_descriptors of the selected program. If several programs on the transport stream are selected, the Host sends all the *ca\_pmt()* APDUs to the Card. The *ca\_pmt()* APDU only contains CA descriptors. All other descriptors SHALL be ignored by the Host.

The CA\_descriptor after the current\_next\_indicator is at the program level and is valid for all elementary components of the program. The CA descriptor(s) at elementary stream level is (are) valid for the elementary



stream only. If, for one elementary stream, CA descriptor(s) exist at program level and at elementary stream level, only the CA descriptors at elementary stream level are taken into account.

The Host SHALL send a new *ca\_pmt()* APDU when:

- the user selects a different program
- the version number changes
- the current next indicator changes

**Table 9.7–5 - S-Mode *ca\_pmt()* APDU Syntax (Resource Type 1 Version 2)**

Syntax	No. of Bits	Mnemonic
<i>ca_pmt()</i> {		
<i>ca_pmt_tag</i>	24	uimsbf
<i>length_field()</i>		
<i>ca_pmt_list_management</i>	8	uimsbf
<i>program_number</i>	16	uimsbf
<i>reserved</i>	2	bslbf
<i>version_number</i>	5	uimsbf
<i>current_next_indicator</i>	1	bslbf
<i>reserved</i>	4	bslbf
<i>program_info_length</i>	12	uimsbf
if ( <i>program_info_length</i> != 0) {		
<i>ca_pmt_cmd_id</i>	8	uimsbf
for ( <i>i</i> =0; <i>i</i> < <i>N</i> ; <i>i</i> ++) {		
CA_descriptor() /* program level */		
}		
}		
for ( <i>i</i> =0; <i>i</i> < <i>N1</i> ; <i>i</i> ++) {		
<i>stream_type</i>	8	uimsbf
<i>reserved</i>	3	bslbf
<i>elementary_PID</i> /* elementary stream PID*/	13	uimsbf
<i>reserved</i>	4	bslbf
<i>ES_info_length</i>	12	uimsbf
if ( <i>ES_info_length</i> != 0) {		
<i>ca_pmt_cmd_id</i> /* at ES level */	8	uimsbf
for ( <i>i</i> =0; <i>i</i> < <i>N2</i> ; <i>i</i> ++) {		
CA_descriptor() /* ES level */		
}		
}		
}		
}		

**ca\_pmt\_tag** 0x9F8032

**ca\_pmt\_list\_management** Indicates whether a single program is selected or a list of multiple programs on the transport stream.

0x00 more – Indicates that the *ca\_pmt* is neither the first nor the last one on the list.

0x01 first – The *ca\_pmt* is the first of a new list of more than one *ca\_pmt()* APDU. All previously selected programs are being replaced by the programs of the new list.

0x02 last – The *ca\_pmt* is the last on the list.

0x03 only – The *ca\_pmt* is the only one on the list.

0x04 add – This *ca\_pmt* is being added to an existing list, that is, a new program has been selected by the user but all previously selected programs

	remain selected. If the <code>program_number</code> has already been received this the action is identical to “update”.
	0x05 update – The <code>ca_pmt</code> is already on the list but either the version number or the <code>current_next_indicator</code> has changed.
	0x06-0xFF Reserved
<b>program_number</b>	The MPEG program number as defined in [ISO13818-1].
<b>version_number</b>	The MPEG version number as defined in [ISO13818-1].
<b>current_next_indicator</b>	The MPEG current/next indicator as defined in [ISO13818-1].
<b>program_info_length</b>	Number of bytes in the <code>program_info</code> field.
<b>ca_pmt_cmd_id</b>	This parameter indicates what response is required from the application to a <i>ca_pmt()</i> APDU. 0x00 Reserved 0x01 ok_descrambling – The Host does not expect an answer to the <code>ca_pmt()</code> APDU and the CableCARD device can start descrambling the program or start an MMI dialog immediately. 0x02 ok_mmi – The application can start a MMI dialog but SHALL NOT start descrambling before reception of a new <code>ca_pmt()</code> APDU with <code>ca_pmt_cmd_id</code> set to “ok_descrambling”. In this case, the Host SHALL guarantee that a MMI session can be opened by the CableCARD device. 0x03 query – The Host expects to receive a <i>ca_pmt_reply()</i> APDU. The Card SHALL NOT start descrambling or start an MMI dialog before reception of a new <code>ca_pmt()</code> APDU with <code>ca_pmt_cmd_id</code> set to either “ok_descrambling” or “ok_mmi”. 0x04 not selected – Indicates to the CableCARD device that the Host no longer requires that the CableCARD device attempt to descramble the service. The CableCARD device SHALL close any MMI dialog it has opened. 0x05-0xFF Reserved
<b>ca_descriptor</b>	The MPEG CA descriptor as defined in [ISO13818-1].
<b>stream_type</b>	The MPEG stream type as defined in [ISO13818-1].
<b>elementary_PID</b>	The MPEG elementary PID as defined in [ISO13818-1].
<b>ES_info_length</b>	Number of bytes in the elementary stream information section.

**Table 9.7-6 - M-Mode *ca\_pmt()* APDU Syntax (Resource Type 2 Version 1)**

Syntax	No. of Bits	Mnemonic
<i>ca_pmt()</i> {		
<i>ca_pmt_tag</i>	24	uimsbf
<i>length_field()</i>		
<i>program_index</i>	8	uimsbf
<i>transaction_id</i>	8	uimsbf
<i>ltsid</i>	8	uimsbf
<i>program_number</i>	16	uimsbf
<i>source_id</i>	16	uimsbf
<i>ca_pmt_cmd_id</i>	8	uimsbf
<i>reserved</i>	4	bslbf
<i>program_info_length</i>	12	uimsbf
if ( <i>program_info_length</i> != 0) {		
CA_descriptor() /* program level */		
}		
for ( <i>i</i> =0; <i>i</i> < <i>N1</i> ; <i>i</i> ++) {		
<i>stream_type</i>	8	uimsbf
<i>reserved</i>	3	bslbf
<i>elementary_PID</i> /* elementary stream PID*/	13	uimsbf
<i>reserved</i>	4	bslbf
<i>ES_info_length</i>	12	uimsbf
if ( <i>ES_info_length</i> != 0) {		
<i>ca_pmt_cmd_id</i> /* at ES level	8	uimsbf
for ( <i>i</i> =0; <i>i</i> < <i>N2</i> ; <i>i</i> ++) {		
CA_descriptor() /* ES level */		
}		
}		
}		
}		

**ca\_pmt\_tag** 0x9F8032

**program\_index** Program index values range from 0 to max\_programs-1, where max\_programs is the value returned by M-CARD in the program\_profile().

**transaction\_id** An 8-bit value, generated by the Host, that will be returned in the corresponding ca\_pmt\_reply() and/or ca\_update() from the M-CARD. The transaction\_id allows the Host to match the M-CARD's replies with the corresponding requests. The Host should increment the value, modulo 255, with every message it sends. A separate transaction\_id counter SHALL be maintained for each program index, so that the transaction\_ids increment independently for each index.

**ltsid** Local Transport Stream ID. Required when the M-CARD is present and operating in Multi-Stream Mode.

**program\_number** The MPEG program number as defined in [ISO13818-1].

**source\_id** The source ID as defined in [SCTE65].

**ca\_pmt\_cmd\_id** This parameter indicates what response is required from the application to a ca\_pmt() APDU.

0x00 Reserved

0x01 ok\_descrambling – The Host does not expect an answer to the ca\_pmt() APDU and the Card can start descrambling the program or start an MMI dialog immediately.

- 0x02 ok\_mmi – The application can start a MMI dialog but SHALL NOT start descrambling before reception of a new *ca\_pmt()* APDU with *ca\_pmt\_cmd\_id* set to “ok\_descrambling”. In this case, the Host SHALL guarantee that a MMI session can be opened by the Card.
- 0x03 query – The Host expects to receive a *ca\_pmt\_reply()* APDU. The Card SHALL NOT start descrambling or start an MMI dialog before reception of a new *ca\_pmt()* APDU with *ca\_pmt\_cmd\_id* set to either “ok\_descrambling” or “ok\_mmi”.
- 0x04 not selected – Indicates to the Card that the Host no longer requires that the Card attempt to descramble the service. The Card SHALL close any MMI dialog it has opened.
- 0x05-0xFF Reserved

**ca\_descriptor**

The MPEG CA descriptor as defined in [ISO13818-1].

**stream\_type**

The MPEG stream type as defined in [ISO13818-1].

**elementary\_PID**

The MPEG elementary PID as defined in [ISO13818-1].

**ES\_info\_length**

Number of bytes in the elementary stream information section.

**9.7.3.1 M-Mode Processing Rules for *ca\_pmt()***

The Host SHALL send a new *ca\_pmt()* APDU when:

- The user selects a different program
- The version number changes
- The current next indicator changes

The *ca\_pmt()* APDU consists of entitlement control information extracted from the Program Map Table (PMT) by the Host and sent to the M-CARD. This control information allows the M-CARD to locate and filter Entitlement Control Message (ECM) streams, and assign the correct ECM stream to each scrambled component.

The *ca\_pmt()* APDU contains CA\_descriptors from the PMT of a selected program. If several programs are selected, then the Host SHALL send a *ca\_pmt()* APDU for each program to the M-CARD. The Host SHALL only include CA descriptors in the *ca\_pmt*, and SHALL NOT include any other descriptors from the PMT.

CA descriptors may be included in the *ca\_pmt()* at the program level and at the elementary stream level. The OCHD2 SHALL apply program level CA descriptors to all elementary streams that have no elementary stream level CA descriptor. When a CA descriptor is present at the elementary stream level, the OCHD2 SHALL apply the CA descriptor to the elementary streams.

The CA descriptors in the PMT are provided by the conditional access system to inform the M-CARD of which PID stream carries the Entitlement Control Messages associated with each program or elementary stream. The Host MAY elect to use some or all of the CA descriptors included in an individual program's PMT to query or request descrambling of that program. However, the CA descriptors in the *ca\_pmt()* SHALL always appear at the same program and elementary stream levels as they originally appeared in the program's PMT. Therefore, program level CA descriptors from a program's PMT SHALL continue to be provided at the program level in the *ca\_pmt()*, and SHALL NOT be replicated at the elementary stream level, even if the Host elects to only request decryption of individual streams from that program.

The *program\_index* tracks the program number and LTSID assigned to a particular CA resource in the M-CARD. The *program\_index* SHALL be carried in the *ca\_pmt()* to allow the M-CARD to maintain a similar index table to track the assignments that it receives from the Host. The assignments to each *program\_index* SHALL be updated as old programs are replaced by new programs, thereby maintaining the total number of active programs within the M-CARD's limitations.

The figure below shows an example of Program Index Table for a Host and M-CARD with the ability to decrypt only two programs. The Program Index Tables track the Host's assignments of the programs that the M-CARD is to decrypt.

**Host Program Index Table**

Program Index	Transaction ID	ItsId	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	19	15	2	1B1	1	51, 52, 53
1	72	90	3	4A8	1	90, 91

**M-CARD Program Index Table**

Program Index	Transaction ID	ItsId	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	19	15	2	1B1	1	51, 52, 53
1	72	90	3	4A8	1	90, 91

**Figure 9.7-1 - Program Index Table 1**

The next figure shows that the Host has made a change to the Program Index 0, to query the M-CARD about a new program. The transaction\_ID is incremented, and the query command is passed through a new *ca\_pmt()*, which updates the M-CARD's Program Index 0. The M-CARD would then reassign its CA resource to evaluate the Host's query, and prepare a *ca\_pmt\_reply()*. The *ca\_pmt\_reply()* would include the same transaction\_id that the Host assigned to the *ca\_pmt()*, and the program index, to uniquely identify the reply.

**Host Program Index Table**

Program Index	Transaction ID	ItsId	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	20	5F	5	8BA	3	100, 101
1	72	90	3	4A8	1	90, 91

**ca-pmt**  
 program\_index = 0  
 transaction\_id = 20  
 ItsId = 5F  
 program\_number = 5  
 source\_id = 8BA  
 ca\_pmt\_cmd\_id = 3

**M-CARD Program Index Table**

Program Index	Transaction ID	ItsId	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	20	5F	5	8BA	3	100, 101
1	72	90	3	4A8	1	90, 91

**Figure 9.7-2 - Program Index Table 2**

The M-CARD SHALL NOT begin descrambling the new program on the basis of a query, even if it replies that descrambling is possible. The final figure shows the Host following up with another *ca\_pmt()*, focused on the same program\_index, with an ok\_descrambling command and new transaction\_ID. The M-CARD will update its index accordingly and begin to descramble the program.

**Host Program Index Table**

Program Index	Transaction ID	Itsid	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	21	5F	5	8BA	1	100, 101
1	72	90	3	4A8	1	90, 91

**ca-pmt**  
 program\_index = 0  
 transaction\_id = 21  
 Itsid = 5F  
 program\_number = 5  
 source\_id = 8BA  
 ca\_pmt\_cmd\_id = 1

**M-CARD Program Index Table**

Program Index	Transaction ID	Itsid	Program Number	Source ID	ca_pmt_cmd_id	Elementary Streams
0	21	5F	5	8BA	1	100, 101
1	72	90	3	4A8	1	90, 91

**Figure 9.7-3 - Program Index Table 3**

The M-CARD SHALL treat the arrival of each *ca\_pmt()* as a new program request, which is either an additional program request, or a replacement for an existing request.

When a *ca\_pmt()* arrives, the M-CARD SHALL allocate the necessary CA resources and assign them to the program index designated in the *ca\_pmt*. If resources have previously been assigned to this program index, then the former allocations SHALL be released and new allocations made to reflect the contents of the new *ca\_pmt()*. Therefore, only a single *ca\_pmt()* will be needed to initiate a complete transition between program assignments and maintain synchronization of resource allocations between the Host and M-CARD.

The Host SHALL assign a *transaction\_id* to each *ca\_pmt()*. The Host SHALL increment the *transaction\_id* with every *ca\_pmt* it sends. The Host SHALL maintain a separate *transaction\_id* counter for each program index, so that the *transaction\_ids* increment independently for each index.

The M-CARD SHALL use the *transaction\_id* from the *ca\_pmt* in any replies or updates it sends regarding that *ca\_pmt()*. This allows the host to match the M-CARD's messages with the corresponding *ca\_pmt()*. Obsolete replies or updates may then be discarded.

In the case where the Host has not yet acquired the Source ID, the Host MAY set this value to 0 until the Source ID information is available, then the Host SHALL replace the previously set value of 0 to that of the received value.

#### 9.7.4 ca\_pmt\_reply

The *ca\_pmt\_reply()* SHALL be sent by the Card to the Host after receiving a *ca\_pmt()* APDU with the *ca\_pmt\_cmd\_id* set to "query". It may also be sent after reception of a CA PMT object with the *ca\_pmt\_cmd\_id* set to 'ok\_mmi' in order to indicate to the Host the result of the MMI dialogue ('descrambling\_possible' if the user has purchased, 'descrambling not possible (because no entitlement)' if the user has not purchased).

**Table 9.7-7 - S-Mode *ca\_pmt\_reply()* APDU Syntax (Resource Type 1 Version 2)**

Syntax	No. of Bits	Mnemonic
<code>ca_pmt_reply() {</code>		
<code>ca_pmt_reply_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>program_number</code>	16	uimsbf
<code>reserved</code>	2	bslbf
<code>version_number</code>	5	uimsbf
<code>current_next_indicator</code>	1	bslbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* program level */</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>for (i=0; i&lt;N1; i++) {</code>		
<code>reserved</code>	3	bslbf
<code>elementary_PID /* elementary stream PID*/</code>	13	uimsbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* elementary stream level*/</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>}</code>		
<code>}</code>		

**ca\_pmt\_reply\_tag**

0x9F8033

**program\_number**

The MPEG program number as defined in [ISO13818-1].

**version\_number**

The MPEG version number as defined in [ISO13818-1].

**current\_next\_indicator**

The MPEG current/next indicator as defined in [ISO13818-1].

**elementary\_PID**

The MPEG elementary PID as defined in [ISO13818-1].

**ca\_enable**Indicates whether the Card is able to perform the descrambling operation requested in the *ca\_pmt()* APDU.

0x00 Reserved

0x01 Descrambling possible with no extra conditions.

0x02 Descrambling possible under conditions (purchase dialog). The Card has to enter a purchase dialog with the user before being able to descramble.

0x03 Descrambling possible under conditions (technical dialog). The Card has to enter a technical dialog with the user before being able to descramble (e.g., request fewer elementary streams because the descrambling capabilities are limited).

0x04-0x70 Reserved

0x71 Descrambling not possible (no entitlement). The selected program is not entitled and is not available for purchase.

0x72 Reserved

0x73 Descrambling not possible (technical reasons); for example, if all the elementary streams capable are being used.

0x74-0xFF Reserved

The syntax contains one possible `ca_enable` at program level and, for each elementary stream, one possible `ca_enable` at elementary stream level.

1. When both are present, only `ca_enable` at ES level applies for that elementary stream
2. When none is present, the Host does not interpret the `ca_pmt_reply` object.

The Card SHALL implement its CA application such that when `ca_enable` is present in `ca_pmt_reply()` both at program level and elementary stream level, only the `ca_enable` at ES level applies for that elementary stream SHALL be applicable to a Card in a network that supports different authorizations at program level and elementary stream level.

**Table 9.7–8 - M-Mode `ca_pmt_reply()` APDU Syntax (Resource Type 2 Version 1)**

Syntax	No. of Bits	Mnemonic
<code>ca_pmt_reply() {</code>		
<code>ca_pmt_reply_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>program_index</code>	8	uimsbf
<code>transaction_id</code>	8	uimsbf
<code>ltsid</code>	8	uimsbf
<code>program_number</code>	16	uimsbf
<code>source_id</code>	16	uimsbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* program level */</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>for (i=0; i&lt;N1; i++) {</code>		
<code>reserved</code>	3	bslbf
<code>elementary_PID /* elementary stream PID*/</code>	13	uimsbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* elementary stream level*/</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>}</code>		
<code>}</code>		

**ca\_pmt\_reply\_tag**

0x9F8033

**program\_index**

The same program index that was used in the original `ca_pmt`. The combination of `transaction_id` and `program_index` uniquely identifies this update.

**transaction\_id**

The same 8-bit `transaction_id` that was used in the original `ca_pmt`.

**ltsid**

Local Transport Stream ID. Required when the M-CARD is present and operating in M-Mode.

**program\_number**

The MPEG program number as defined in [ISO13818-1].

**source\_id**

The source ID as defined in [SCTE65].

**elementary\_PID**

The MPEG elementary PID as defined in [ISO13818-1].

**ca\_enable**

Indicates whether the Card is able to perform the descrambling operation requested in the `ca_pmt()` APDU.



- 0x00 Reserved
- 0x01 Descrambling possible with no extra conditions.
- 0x02 Descrambling possible under conditions (purchase dialog). The Card has to enter a purchase dialog with the user before being able to descramble.
- 0x03 Descrambling possible under conditions (technical dialog). The Card has to enter a technical dialog with the user before being able to descramble (e.g., request fewer elementary streams because the descrambling capabilities are limited).
- 0x04-0x70 Reserved
- 0x71 Descrambling not possible (no entitlement). The selected program is not entitled and is not available for purchase.
- 0x72 Reserved
- 0x73 Descrambling not possible (technical reasons), for example if all the elementary streams capable are being used.
- 0x74-0xFF Reserved

### 9.7.5 ca\_update

The Card SHALL use the *ca\_update()* APDU to inform the Host when CA information for the currently tuned program has changed. Note that *ca\_update()* SHALL always reference the service to which the Host is currently tuned. This is the last service for which a *ca\_pmt()* APDU was sent from the Host to the Card that was not a query.

**Table 9.7–9 - S-Mode ca\_update() APDU Syntax (Resource Type 1 Version 2)**

Syntax	No. of Bits	Mnemonic
<code>ca_update() {</code>		
<code>ca_update_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>program_number</code>	16	uimsbf
<code>reserved</code>	2	bslbf
<code>version_number</code>	5	uimsbf
<code>current_next_indicator</code>	1	bslbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* program level */</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>for (i=0; i&lt;N1; i++) {</code>		
<code>reserved</code>	3	bslbf
<code>elementary_PID /* elementary stream PID*/</code>	13	uimsbf
<code>CA_enable_flag</code>	1	bslbf
<code>if (CA_enable_flag == 1) {</code>		
<code>CA_enable /* elementary stream level*/</code>	7	uimsbf
<code>}</code>		
<code>else {</code>		
<code>reserved</code>	7	uimsbf
<code>}</code>		
<code>}</code>		
<code>}</code>		

**ca\_update\_tag** 0x9F8034

**program\_number** The MPEG program number as defined in [ISO13818-1].

**version\_number** The MPEG version number as defined in [ISO13818-1].

<b>current_next_indicator</b>	The MPEG current/next indicator as defined in [ISO13818-1].
<b>elementary_PID</b>	The MPEG elementary PID as defined in [ISO13818-1].
<b>ca_enable</b>	Indicates whether the Card is able to perform the descrambling operation requested in the <i>ca_pmt()</i> APDU. 0x00 Reserved 0x01 Descrambling possible with no extra conditions. 0x02 Descrambling possible under conditions (purchase dialog). The Card has to enter a purchase dialog with the user before being able to descramble. 0x03 Descrambling possible under conditions (technical dialog). The Card has to enter a technical dialog with the user before being able to descramble (e.g., request fewer elementary streams because the descrambling capabilities are limited). 0x04-0x70 Reserved 0x71 Descrambling not possible (no entitlement). The selected program is not entitled and is not available for purchase. 0x72 Reserved 0x73 Descrambling not possible (technical reasons), for example if all the elementary streams capable are being used. 0x74-0xFF Reserved

**Table 9.7–10 - M-Mode ca\_update() APDU Syntax (Resource Type 2 Version 1)**

Syntax	No. of Bits	Mnemonic
ca_update() {		
ca_update_tag	24	uimsbf
length_field()		
program_index	8	uimsbf
transaction_id	8	uimsbf
ltsid	8	uimsbf
program_number	16	uimsbf
source_id	16	uimsbf
CA_enable_flag	1	bslbf
if (CA_enable_flag == 1) {		
CA_enable /* program level */	7	uimsbf
}		
else {		
reserved	7	uimsbf
}		
for (i=0; i<N1; i++) {		
reserved	3	bslbf
elementary_PID /* elementary stream PID*/	13	uimsbf
CA_enable_flag	1	bslbf
if (CA_enable_flag == 1) {		
CA_enable /* elementary stream level*/	7	uimsbf
}		
else {		
reserved	7	uimsbf
}		
}		
}		

**ca\_update\_tag** 0x9F8034

**program\_index** The same program index that was used in the original ca\_pmt. The combination of transaction\_id and program\_index uniquely identifies this update.

<b>transaction_id</b>	The same 8-bit transaction_id that was used in the original ca_pmt.
<b>ltsid</b>	Local Transport Stream ID. Required when the M-CARD is present, and operating in M-Mode.
<b>program_number</b>	The MPEG program number as defined in [ISO13818-1].
<b>source_id</b>	The source ID as defined in [SCTE65].
<b>elementary_PID</b>	The MPEG elementary PID as defined in [ISO13818-1].
<b>ca_enable</b>	Indicates whether the Card is able to perform the descrambling operation requested in the <i>ca_pmt()</i> APDU. 0x00 Reserved 0x01 Descrambling possible with no extra conditions. 0x02 Descrambling possible under conditions (purchase dialog). The Card has to enter a purchase dialog with the user before being able to descramble. 0x03 Descrambling possible under conditions (technical dialog). The Card has to enter a technical dialog with the user before being able to descramble (e.g., request fewer elementary streams because the descrambling capabilities are limited). 0x04-0x70 Reserved 0x71 Descrambling not possible (no entitlement). The selected program is not entitled and is not available for purchase. 0x72 Reserved 0x73 Descrambling not possible (technical reasons); for example, if all the elementary streams capable are being used. 0x74-0xFF Reserved

The different APDU tag prevents any confusion between *ca\_pmt\_reply()* and *ca\_update()* APDUs during *ca\_pmt(query)/ca\_pmt\_reply()* exchanges. The *ca\_pmt(query)* is sent by the Host to determine which conditional access resource can decrypt the specified service *when more than one conditional access resource is present*. The reader should note that some conditional access implementations may send a CA\_PMT (query) each time a service is tuned and with only one Card installed in the Host. This is done to determine if the currently tuned service can be descrambled by the Card. The Card may respond with a *ca\_pmt\_reply()* specifying “descrambling possible” or “descrambling not possible”. The Host would respond to a *ca\_pmt\_reply()* (descrambling\_possible) with a CA\_PMT (ok\_descrambling) to the Card.

The syntax contains one possible ca\_enable at program level and, for each elementary stream, one possible ca\_enable at elementary stream level.

When both are present, only ca\_enable at ES level applies for that elementary stream.

When none is present, the Host does not interpret the *ca\_pmt\_reply()* object.

## 9.8 Host Control

This resource allows the Card to set up the Host OOB RF receiver and transmitter, if available, and to allow the Card the capability to tune the Host’s FAT tuner under certain conditions defined by the Homing resource. Unless a session to the Homing resource has been opened and the Card granted access to the FAT tuner, or if the Host is in the standby state and allows the Card access to the tuner (the Host MAY or MAY NOT allow this), the Host SHOULD NOT grant any request by the Card to tune the FAT tuner. Only one session SHALL be opened. This M-CARD Resource operating in M-Mode has been modified from a Card operating in the S-Mode. The resource Type value has been changed to indicate the modified resource.

**Table 9.8–1 - Host Control Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Host Control	S-Mode	32	1	3	0x00200043
Host Control	M-Mode	32	2	1	0x00200081

The Card will have the Host Control resource open for control of the OOB receiver and transmitter and SHALL leave it open independent of the operation of the Homing resource.

The creation of the specification application resource includes the following objects:

The Host Control resource consists of 6 APDUs.

**Table 9.8–2 - Host Control Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
OOB_TX_tune_req()	0x9F8404	Host Control	←
OOB_TX_tune_cnf()	0x9F8405	Host Control	→
OOB_RX_tune_req()	0x9F8406	Host Control	←
OOB_RX_tune_cnf()	0x9F8407	Host Control	→
inband_tune_req()	0x9F8408	Host Control	←
inband_tune_cnf()	0x9F8409	Host Control	→

### 9.8.1 OOB\_TX\_tune\_req

If the DSG option is not selected, the Card SHALL use the *OOB\_TX\_tune\_req()* APDU to set up the Host's RF transmitter.

**Table 9.8–3 - OOB\_TX\_tune\_req() APDU Syntax**

Syntax	No. of Bits	Mnemonic
OOB_TX_tune_req() { OOB_TX_tune_req_tag Length_field() RF_TX_frequency_value RF_TX_power_level RF_TX_rate_value }	24   16 8 8	uimsbf   uimsbf uimsbf uimsbf

**OOB\_TX\_tune\_req\_tag** 0x9F8404

**RF\_TX\_frequency\_value** This field defines the frequency of the RF transmitter, in kHz.

**RF\_TX\_power\_level** This value defines the power level of the RF transmitter, in units of 0.5 dBmV. The value 0x00 SHALL correspond to an output level of 0 dBmV.

**RF\_TX\_rate\_value** This value defines the bit rate of the RF transmitter. The format and values are defined in Table 9.8–6.

**Table 9.8–4 - RF TX Frequency Value**

Bit	7	6	5	4	3	2	1	0
	Frequency (MS)							
	Frequency (LS)							

**RF\_TX\_frequency\_value** This field defines the frequency of the RF Transmitter, in kHz.

**Table 9.8–5 - RF TX Power Level**

Bit	7	6	5	4	3	2	1	0
	RF Power Level							

**RF\_TX\_power\_level**

Power level of the RF Transmitter, in units of 0.5dBmV. The value 0x00 SHALL correspond to an output level of 0 dBmV.

**Table 9.8–6 - RF TX Rate Value**

Bit	7	6	5	4	3	2	1	0
	Rate		Reserved					

**RF\_TX\_rate\_value**

**Rate** – Bit rate.

00b = 256 kbps

01b = 2048 kbps

10b = 1544 kbps

11b = 3088 kbps

**9.8.2 OOB\_TX\_tune\_cnf**

Upon reception of an *OOB\_TX\_tune\_req()* APDU and tuning the transmitter, the Host SHALL send the *OOB\_TX\_tune\_cnf()* APDU to the Card.

**Table 9.8–7 - OOB\_TX\_tune\_cnf() APDU Syntax**

Syntax	No. of Bits	Mnemonic
OOB_TX_tune_cnf() {		
OOB_TX_tune_cnf_tag	24	uimsbf
length_field()		
status_field	8	uimsbf
}		

**OOB\_TX\_tune\_cnf\_tag**

0x9F8405

**status\_field**

This field returns the status of the *OOB\_TX\_tune\_req()*. If the request was granted and the RF Transmitter set to the desired configuration, *Status\_field* will be set to 0x00. If the Host is a unidirectional Host, *Status\_field* SHALL be set to 0x01; the Card SHALL NOT attempt to perform RF transmit operations after receiving an *OOB\_TX\_tune\_cnf()* with *Status\_field* set to 0x01. If any of the parameters passed to the Host are outside of the Host's tuning rate, then the Host SHALL transmit the *OOB\_TX\_tune\_cnf()* with *Status\_field* set to 0x03. Otherwise *Status\_field* will be set to one of the following values:

0x00 Tuning granted

0x01 Tuning denied – RF transmitter no physically available

0x02 Tuning denied – RF transmitter busy

0x03 Tuning denied – Invalid parameters

0x04 Tuning denied – Other reasons

0x05-0xFF Reserved

**9.8.3 OOB\_RX\_tune\_req**

If the DSG option is not selected, the Card SHALL use the *OOB\_RX\_tune\_req()* APDU to set up the Host's RF receiver.

**Table 9.8–8 - OOB\_RX\_tune\_req() APDU Syntax**

Syntax	No. of Bits	Mnemonic
OOB_RX_tune_req() { OOB_RX_tune_req_tag length_field() RF_RX_frequency_value RF_RX_data_rate }	24  16 8	uimsbf  uimsbf uimsbf

**OOB\_RX\_tune\_req\_tag** 0x9F8406

**RF\_RX\_frequency\_value** This field defines the frequency of the RF receiver.  
(Frequency = value \* 0.05 + 50 MHz.). The format is defined in Table 9.8–9.

**Table 9.8–9 - RF RX Frequency Value**

Bit	7	6	5	4	3	2	1	0
	0	0	0	0	0	Value (MS)		
	Value (LS)							

**RF\_RX\_data\_rate** This value defines the bit rate and spectral inversion of the RF transmitter. The format is defined in Table 9.8–10.

The following is the definition of the bit rate.

00b 2,048 kbps  
01b 2,048 kbps  
10b 1,544 kbps  
11b 3,088 kbps

The following is the definition of the Spectral Inversion (SPEC).

0 Spectrum is non-inverted  
1 Spectrum is inverted

**Table 9.8–10 - OOB Transmit Rate Format**

Bit	7	6	5	4	3	2	1	0
	Rate							SPEC

#### 9.8.4 OOB\_RX\_tune\_cnf

Upon reception of an **OOB\_RX\_tune\_req()** APDU and tuning the RF receiver, the Host SHALL send the **OOB\_RX\_tune\_cnf()** APDU to the Card. The **OOB\_RX\_tune\_cnf()** APDU SHALL only be transmitted after either the requested frequency has been tuned and acquired (“tune time”), or 500 msec has elapsed since receiving the request, whichever comes first.

**Table 9.8–11 - OOB\_RX\_tune\_cnf() APDU Syntax**

Syntax	No. of Bits	Mnemonic
OOB_RX_tune_cnf() { OOB_RX_tune_cnf_tag length_field() status_field }	24  8	uimsbf  uimsbf

**OOB\_RX\_tune\_cnf\_tag** 0x9F8407

**status\_field** Returns the status of the RF receiver. The following values are to be used:

0x00 Tuning granted  
 0x01 Tuning denied – RF receiver not physically available  
 0x02 Tuning denied – RF receiver busy  
 0x03 Tuning denied – Invalid parameters  
 0x04 Tuning denied – Other reasons  
 0x05-0xFF Reserved

### 9.8.5 inband\_tune\_req

The *inband\_tune\_req()* APDU allows for the Card to request the Host to tune the inband QAM tuner. The APDU will allow support for tuning to a source\_id or a frequency with the modulation type.

**Table 9.8–12 - S-Mode - inband\_tune\_req() APDU Syntax (Resource Type 1 Version 3)**

Syntax	No. of Bits	Mnemonic
inband_tune_req() {		
inband_tune_req_tag	24	uimsbf
length_field()		
tune_type	8	uimsbf
if (tune_type == 0x00) {		
source_id	16	uimsbf
}		
else if (tune_type == 0x01) {		
tune_frequency_value	16	uimsbf
modulation_value	8	uimsbf
}		
}		

**inband\_tune\_req\_tag**

0x9F8408

**tune\_type**

Determines whether to use the source ID value or the frequency and modulation values.

0x00 Source ID  
 0x01 Frequency  
 0x02-0xFF Reserved

**source\_id**

When tune\_type = 0x00, the source\_id is a 16 bit unsigned integer in the range of 0x0000 to 0xFFFF that identifies the programming source associated with the virtual channel on a system wide basis. In this context, a source is one specific source of video, text, data, or audio programming. For the purposes of referencing virtual channels to the program guide database, each such program source is associated with a unique value of source\_id. The source\_id itself may appear in an IPG database, where it tags entries to associate them with specific services. The value zero for source\_id, if used, SHALL indicate the channel is not associated with a source\_id.

**Tune\_frequency\_value**

When tune\_type = 0x01, tune\_frequency\_value contains the frequency for the Host to tune. The frequency is calculated by multiplying tune\_frequency\_value by 0x0.05 MHz (50 kHz resolution). The format is defined in Table 9.8–13.

**Table 9.8–13 - Tune Frequency Value**

Bit	7	6	5	4	3	2	1	0
MSB	Value (MS)							
LSB	Value (LS)							

**modulation\_value**

When tune\_type = 0x01, modulation value sets the type of modulation for the inband tuner.

0x00 64QAM  
 0x01 256QAM  
 0x02-0xFF Reserved

**Table 9.8–14 - M-Mode - inband\_tune\_req() APDU Syntax (Resource Type 2 Version 1)**

Syntax	No. of Bits	Mnemonic
inband_tune_req() { inband_tune_req_tag length_field()  tune_type if (tune_type == 0x00) { source_id } else if (tune_type == 0x01) { tune_frequency_value modulation_value } }	24   8 16 16 8	uimsbf   uimsbf uimsbf uimsbf uimsbf

**inband\_tune\_req\_tag** 0x9F8408

**Tune\_type** Determines whether to use the source ID value or the frequency and modulation values.

0x00 Source ID  
 0x01 Frequency  
 0x02-0xFF Reserved

**source\_id** When tune\_type = 0x00, the source\_id is a 16 bit unsigned integer in the range of 0x0000 to 0xFFFF that identifies the programming source associated with the virtual channel on a system wide basis. In this context, a source is one specific source of video, text, data, or audio programming. For the purposes of referencing virtual channels to the program guide database, each such program source is associated with a unique value of source\_id. The source\_id itself may appear in an IPG database, where it tags entries to associate them with specific services. The value zero for source\_id, if used, SHALL indicate the channel is not associated with a source\_id.

**Tune\_frequency\_value** When tune\_type = 0x01, tune\_frequency\_value contains the frequency for the Host to tune. The frequency is calculated by multiplying tune\_frequency\_value by 0x05 MHz (50 kHz resolution). The format is defined in Table 9.8–15.

**Table 9.8–15 - Tune Frequency Value**

Bit	7	6	5	4	3	2	1	0
MSB	Value (MS)							
LSB	Value (LS)							

**modulation\_value** When tune\_type = 0x01, modulation value sets the type of modulation for the inband tuner.

0x00 64QAM  
 0x01 256QAM  
 0x02-0xFF Reserved



### 9.8.6 inband\_tune\_cnf

When the Host receives an *inband\_tune\_req()* APDU, it SHALL respond with the following APDU.

**Table 9.8–16 - S-Mode - inband\_tune\_cnf() APDU Syntax (Resource Type 1 Version 3)**

Syntax	No. of Bits	Mnemonic
inband_tuning_cnf() { inband_tuning_cnf_tag	24	uimsbf
length_field() tune_status }	8	uimsbf

**inband\_tuning\_cnf\_tag** 0x9F8409

**tune\_status** The Host's response to the *inband\_tuning\_req()* APDU.

- 0x00 Tuning accepted
- 0x01 Invalid frequency (Host does not support this frequency)
- 0x02 Invalid modulation (Host does not support this modulation type)
- 0x03 Hardware failure (Host has hardware failure)
- 0x04 Tuner busy (Host is not relinquishing control of inband tuner)
- 0x05-0xFF Reserved

**Table 9.8–17 - M-Mode - inband\_tune\_cnf() APDU Syntax (Resource Type 1 Version 3)**

Syntax	No. of Bits	Mnemonic
inband_tuning_cnf() { inband_tuning_cnf_tag	24	uimsbf
length_field() ltsid	8	uimsbf
tune_status }	8	uimsbf

**inband\_tuning\_cnf\_tag** 0x9F8409

**ltsid** Local Transport Stream ID. Utilized when the M-CARD is present, and operating in M-Mode.

**Tune\_status** The Host's response to the *inband\_tuning\_req()* APDU.

- 0x00 Tuning accepted
- 0x01 Invalid frequency (Host does not support this frequency)
- 0x02 Invalid modulation (Host does not support this modulation type)
- 0x03 Hardware failure (Host has hardware failure)
- 0x04 Tuner busy (Host is not relinquishing control of inband tuner)
- 0x05-0xFF Reserved

## 9.9 Generic IPPV Support

The CableCARD Interface support of the Generic IPPV resource is deprecated. If supported, it SHALL comply with Section 8.10 of ANSI/SCTE 28.

## 9.10 System Time

The system time resource is supplied by the Host and only one session is supported. The Card creates a session to the resource and then inquires the current time with a *system\_time\_inq()* APDU. If response\_interval is zero, the response is a single *system\_time()* APDU immediately. If response\_interval is non-zero, the response is a *system\_time()* APDU, immediately followed by further *system\_time()* APDUs, every response\_interval seconds.

This resource has been modified from the one in [NRSSB]. The Type value has been changed to indicate the modified resource.

**Table 9.10–1 - System Time Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
System Time	S-Mode/M-Mode	36	1	1	0x00240041
System Time	S-Mode/M-Mode	36	1	2	0x00240042

The System Time resource consists of 2 APDUs.

**Table 9.10–2 - System Time Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
system_time_inq	0x9F8442	System Time	←
system_time	0x9F8443	System Time	→

### 9.10.1 system\_time\_inq

The Card transmits the system\_time\_inq to the Host.

**Table 9.10–3 - Transmission of system\_time\_inq**

Syntax	No. of Bits	Mnemonic
system_time_inq () { system_time_inq_tag length_field() response_interval }	24	uimsbf
	8	uimsbf

**system\_time\_inq\_tag**                      0x9F8442

**response\_interval**                      How often, in seconds, the Host SHOULD transmit the *system\_time()* APDU to the Card, starting immediately. A value of 0x00 means that only a single *system\_time()* APDU is to be transmitted, immediately.

### 9.10.2 system\_time

The *system\_time()* APDU is transmitted from the Host to the Card in response to the *system\_time\_inq()* APDU and then, for non-zero response\_interval values, every response\_interval seconds.

**Table 9.10–4 - system\_time APDU (Type 1)**

Syntax	No. of Bits	Mnemonic
system_time() { system_time_tag length_field() system_time GPS_UTC_offset	24	uimsbf
	32	uimsbf
	8	uimsbf

**Table 9.10–5 - system\_time APDU (Type 2)**

Syntax	No. of Bits	Mnemonic
system_time() {		
system_time_tag	24	uimsbf
length_field()		
system_time_value	32	uimsbf
GPS.UTC_offset	8	uimsbf
DST_Info	2	Uimsbf
DS_Status	1	Uimsbf
DS_day	5	Uimsbf
DS_hour	8	uimsbf
}		

**system\_time\_tag** 0x9F8443

**system\_time\_value** A 32-bit unsigned integer quantity representing the current system time as the number of seconds since 12 AM, January 6, 1980.

**GPS.UTC\_offset** An 8-bit unsigned integer that defines the current offset in whole seconds between GPS and UTC time standards. To convert GPS time to UTC, the GPS.UTC\_offset is subtracted from GPS time. Whenever the International Bureau of Weights and Measures decides that the current offset is too far in error, an additional leap second MAY be added (or subtracted), and the GPS.UTC\_offset will reflect that change.

**DST\_Info** Indicates the status of the following Daylight Savings information:

- 00 None of the following DS information is valid
- 01 The DS\_Status flag is valid but the DS\_day and DS\_hour is not.
- 11 The DS\_Status, DS\_day and DS\_hour information are all valid.

**DS\_Status** Indicates the status of Daylight Savings:

- 0 Not in daylight savings time
- 1 In daylight savings time

**DS\_day** The local day of the month on which the transition to/from DST occurs (1-31)

**DS\_hour** The local hour at which transition to/from DST occurs (0-23)

## 9.11 Man-Machine Interface (MMI)

The Man-Machine Interface (MMI) resource resides in the Host. The Card SHALL only open one session to this resource if it wants to initialize one or more MMI dialogs. This session SHALL remain open during normal operation.

The MMI resource provides the following:

- Support to the Card to open an MMI dialog
- Support to the Host to confirm that the MMI dialog has been opened
- Support to the Card to close the MMI dialog it opened
- Support to the Host to confirm that the MMI dialog has been closed either upon Host or Card request

For S-Mode, the Man-Machine Interface resource has been changed to type 2 to reflect the changes listed into this section compared to EIA-679-B.

For M-Mode the MMI resource has been changed to type 3 to reflect the changes listed into this section compared to Type and Version for the S-Mode MMI resource.

The MMI resource has been updated to allow for implementing on M-Hosts, using the transport stream ID. This allows for the Card to determine which transport stream SHOULD display the desired MMI dialog. With the addition of this new functionality, there are circumstances in which the Card MAY choose to display the MMI screen on 1) all of the Host outputs, 2) only non-recorded Host outputs, 3) the display (for Host with displays, i.e., televisions), and 4) specific transport streams.

**Table 9.11–1 - MMI Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
MMI	S-Mode	64	2	1	0x00400081
MMI	M-Mode	64	3	1	0x004000C1

The MMI resource consists of 4 APDUs.

**Table 9.11–2 - MMI Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
open_mmi_req()	0x9F8820	MMI	←
open_mmi_cnf()	0x9F8821	MMI	→
close_mmi_req()	0x9F8822	MMI	←
close_mmi_cnf()	0x9F8823	MMI	→

### 9.11.1 open\_mmi\_req

The Card SHALL send an *open\_mmi\_req()* APDU to the Host when it wants to initialize an MMI dialog. For a Host that supports more than one MMI dialog at the same time (multiple windows), the Card MAY send another *open\_mmi\_req()* APDU before it closes the previous one.

**Table 9.11–3 - open\_mmi\_req()**

Syntax	No. of Bits	Mnemonic
open_mmi_req() { open_mmi_req_tag length_field() display_type url_length for (i=0; i<url_length; i++) { url_byte } }	24  8 16 8	uimsbf  uimsbf uimsbf uimsbf

**open\_mmi\_req\_tag**

0x9F8820

**display\_type**

Describes how the MMI dialog SHOULD take place. For a Host that supports more than one MMI dialog at the same time, the new MMI dialog can be in the current window or in a new one. One resource class is provided. It supports display and keypad instructions to the user. Note that the Host indicates to the Card which Display Types it supports via the Application Info Resource.

0x00 Full screen  
0x01 Overlay  
0x02 New window  
0x03-0xFF Reserved

**url\_length** Number of bytes in the following loop.

**url\_byte** Each url\_byte is one octet of a parameter that points to a HTML page in the Card and that needs to be queried by the Host using the *server\_query()* APDU (Application Info resource) when the MMI dialog is opened.

### 9.11.2 open\_mmi\_cnf

After receiving an *open\_mmi\_req()* APDU from the Card operating in S-Mode, the Host SHALL reply with an *open\_mmi\_cnf()* APDU to confirm the status of the request.

**Table 9.11–4 - S-Mode - open\_mmi\_cnf (Resource Type 2)**

Syntax	No. of Bits	Mnemonic
<code>open_mmi_cnf() {</code>		
<code>open_mmi_cnf_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>dialog_number</code>	8	uimsbf
<code>open_status</code>	8	uimsbf
<code>}</code>		

**open\_mmi\_cnf\_tag** 0x9F8821

**dialog\_number** A number supplied by the Host issued from an 8-bit cyclic counter that uniquely identifies each *open\_mmi\_cnf()* APDU and allows the Card to close the associated MMI dialog.

**open\_status** The status of the requested MMI dialog defined as follows:

0x00 OK- Dialog opened  
 0x01 Request denied – Host busy  
 0x02 Request denied – Display type not supported  
 0x03 Request denied – No video signal  
 0x04 Request denied – No more windows available  
 0x05-0xFF Reserved

After receiving an *open\_mmi\_req()* APDU from the Card operating in M-Mode, the Host SHALL reply with an *open\_mmi\_cnf()* APDU to confirm the status of the request.

**Table 9.11–5 - M-Mode - open\_mmi\_cnf (Resource Type 3)**

Syntax	No. of Bits	Mnemonic
<code>open_mmi_cnf() {</code>		
<code>open_mmi_cnf_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>display_stream_type</code>	8	uimsbf
<code>if (stream_type == 0x03) {</code>		
<code>ltsid</code>	8	uimsbf
<code>}</code>		
<code>dialog_number</code>	8	uimsbf
<code>open_status</code>	8	uimsbf
<code>}</code>		

**open\_mmi\_cnf\_tag** 0x9F8821

**display\_stream\_type** Type of stream display

0x00 All transport streams  
 0x01 Only on non-recording transport streams

- 0x02 Only on display (for Hosts with display capability)
- 0x03 Only on a specific transport stream
- 0x04-0xFF Reserved

**itsid** Local Transport Stream ID. Required when the M-CARD is present, and operating in M-Mode and the stream\_type = 0x03.

**dialog\_number** A number supplied by the Host issued from an 8-bit cyclic counter that uniquely identifies each *open\_mmi\_cnf()* APDU and allows the Card to close the associated MMI dialog.

**open\_status** The status of the requested MMI dialog defined as follows:

- 0x00 OK- Dialog opened
- 0x01 Request denied – Host busy
- 0x02 Request denied – Display type not supported
- 0x03 Request denied – No video signal
- 0x04 Request denied – No more windows available
- 0x05-0xFF Reserved

### 9.11.3 close\_mmi\_req

The Card operating in S-Mode or in M-Mode SHALL send a *close\_mmi\_req()* APDU to the Host to close a MMI dialog previously opened with an *open\_mmi\_req()* APDU.

**Table 9.11–6 - close\_mmi\_req**

Syntax	No. of Bits	Mnemonic
<i>close_mmi_req()</i> { <i>close_mmi_req_tag</i> <i>length_field()</i> <i>dialog_number</i> }	24	uimsbf
	8	uimsbf

**close\_mmi\_req\_tag** 0x9F8822

**dialog\_number** The number of the MMI dialog assigned by the Host in the *open\_mmi\_cnf()* APDU.

### 9.11.4 close\_mmi\_cnf

After receiving a *close\_mmi\_req()* APDU from the Card, the Host SHALL reply with a *close\_mmi\_cnf()* APDU to confirm the status of the close operation. The Host MAY send a *close\_mmi\_cnf()* APDU without the Card having sent a *close\_mmi\_req()* APDU to inform the Card about a close operation performed by the Host (e.g., the subscriber closes the window).

**Table 9.11–7 - close\_mmi\_cnf**

Syntax	No. of Bits	Mnemonic
<i>close_mmi_cnf()</i> { <i>close_mmi_cnf_tag</i> <i>length_field()</i> <i>dialog_number</i> }	24	uimsbf
	8	uimsbf

**close\_mmi\_cnf\_tag** 0x9F8823

**dialog\_number** The number of the MMI dialog received in the *close\_mmi\_req()* APDU.

## 9.12 M-Mode Device Capability Discovery

The M-CARD when operating in M-Mode SHALL indicate to the Host what its multi-stream capabilities are. It SHALL communicate the maximum number of transport streams it supports via the *stream\_profile()* APDU, how many programs via *program\_profile()* APDU, and how many elementary streams via *es\_profile()* APDU. The Host SHALL reply in each case with an appropriate confirmation APDU.

The maximum number of elementary streams does not include those PIDs which are consumed internally by the Card for internal Card applications. This number refers only to PIDs of programs that are consumed by the Host for viewing or storage. If the Card requires additional PIDs for internal consumption, they are not to be included in this number.

The Card SHALL inform the Host which PIDs it requires through the CableCARD Capability Discovery resource, CARD RES, and the Host SHALL NOT remove those PIDs from each transport stream. Since the PIDs that the Card requires MAY change, the Host SHALL be ready to receive the *request\_pids\_cnf()* APDU at any time and the Host SHALL respond accordingly to those updates. The Card MAY request a maximum of 8 PIDs be transmitted to it per program.

In the event that the Card is unable to meet the stream, program, or PID processing requirements that a user has requested, it SHALL be the Host's responsibility to notify the user.

When Operating in M-Mode, the Host is responsible for controlling the data through the CHI such that the data rate does not exceed the interface maximum. The Host MAY perform this by removing selected packets from the outgoing multiplexed transport stream with PIDs that are not used in order to have sufficient bandwidth for the CHI.

**Table 9.12–1 - CableCARD Device Resources Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
CARD RES	M-Mode	38	3	1	0x002600C1

The Card Resources resource consists of 8 APDUs.

**Table 9.12–2 - CableCARD Resources Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ Card
stream_profile()	0x9FA010	CARD RES	←
stream_profile_cnf()	0x9FA011	CARD RES	→
program_profile()	0x9FA012	CARD RES	←
program_profile_cnf()	0x9FA013	CARD RES	→
es_profile()	0x9FA014	CARD RES	←
es_profile_cnf()	0x9FA015	CARD RES	→
request_pids()	0x9FA016	CARD RES	→
request_pids_cnf()	0x9FA017	CARD RES	←

### 9.12.1 stream\_profile APDU

After the Card capability discovery session is established, the Card SHALL transmit the *stream\_profile()* APDU to the Host. This includes the maximum number of streams that the Card can support.

**Table 9.12–3 - stream\_profile APDU Syntax**

Syntax	# of bits	Mnemonic
stream_profile() { stream_profile_tag	24	uimsbf
length_field() max_number_of_streams }	8	uimsbf

**stream\_profile\_tag** Value = 0x9FA010

**max\_number\_of\_streams** The maximum number of unique MPEG transport streams input into the Card from the Host that the Card Operating in M-Mode can manage. The value SHALL be greater than three.

### 9.12.2 stream\_profile\_cnf APDU

When the Host receives the *stream\_profile()* APDU from the Card, it SHALL respond with the following APDU.

**Table 9.12–4 - stream\_profile\_cnf APDU**

Syntax	# of bits	Mnemonic
stream_profile_cnf() { stream_profile_cnf_tag	24	uimsbf
length_field() number_of_streams_used }	8	uimsbf

**stream\_profile\_cnf\_tag** Value = 0x9FA011

**number\_of\_streams\_used** The number of unique MPEG transport streams that the Host will be sending to the Card simultaneously.

### 9.12.3 program\_profile APDU

After the Card capability discovery session is established, the Card SHALL transmit the *program\_profile()* APDU to the Host. This includes the maximum number of simultaneous programs, summed across all transport streams that the Card's CA system can simultaneously decrypt.

**Table 9.12–5 - program\_profile APDU**

Syntax	# of bits	Mnemonic
program_profile() { program_profile_tag	24	uimsbf
length_field() max_number_of_programs }	8	uimsbf

**program\_profile\_tag** Value = 0x9FA012

**max\_number\_of\_programs** The maximum number of programs that the Card's CA system can simultaneously decrypt SHALL be greater than or equal to four.



### 9.12.4 programs\_profile\_cnf APDU

When the Host receives the *program\_profile()* APDU from the Card, it SHALL respond with the following APDU.

**Table 9.12–6 - programs\_profile\_cnf APDU**

Syntax	# of bits	Mnemonic
<pre>program_profile_cnf() {     program_profile_cnf_tag     length_field() }</pre>	24	uimsbf

**program\_profile\_cnf\_tag** Value = 0x9FA013

### 9.12.5 es\_profile APDU

After the Card capability discovery session is established, the Card SHALL transmit the *es\_profile()* APDU to the Host. This includes the maximum number of simultaneous elementary streams, summed across all transport streams that the Card can support.

This maximum number does not include those PIDs which are consumed internally by the Card for internal Card applications. This number refers only to PIDs of programs that are consumed by the Host for viewing or storage. If the Card requires additional PIDs for internal consumption, they are not to be included in this number.

**Table 9.12–7 - es\_profile APDU Syntax**

Syntax	# of bits	Mnemonic
<pre>es_profile() {     es_profile_tag     length_field()     max_number_of_es }</pre>	24	uimsbf
	8	uimsbf

**es\_profile\_tag** Value = 0x9FA014

**max\_number\_of\_es** The maximum number of elementary streams that the Card operating in M-Mode can manage SHALL be greater than or equal to sixteen (16).

### 9.12.6 es\_profile\_cnf APDU

When the Host receives the *es\_profile()* APDU from the Card, it SHALL respond with the following APDU.

**Table 9.12–8 - es\_profile\_cnf APDU Syntax**

Syntax	# of bits	Mnemonic
<pre>es_profile_cnf() {     es_profile_cnf_tag     length_field() }</pre>	24	uimsbf

**es\_profile\_cnf\_tag** Value = 0x9FA015

### 9.12.7 request\_pids APDU

If the Host has to perform filtering (removing transport packets corresponding to PIDs that it does not need) to keep the bandwidth below the maximum of the interface, the Host SHALL ask the Card for the PIDs that it must receive

and therefore the Host SHALL NOT filter. The *request\_pids()* APDU SHALL be used by the Host to request a list of PIDs that SHALL NOT be filtered.

**Table 9.12–9 - request\_pids APDU**

Syntax	# of bits	Mnemonic
request_pids() { request_pids_tag length_field() ltsid pid_filtering_status }	24  8 8	uimsbf  uimsbf uimsbf

**request\_pids\_tag** Value = 0x9FA016

**ltsid** Local Transport Stream ID. Only required when the M-CARD is present and operating in M-Mode.

**pid\_filtering\_status**  
    0x00 Host not filtering PIDs  
    0x01 Host filtering PIDs  
    0x02-FF Reserved

### 9.12.8 request\_pids\_cnf APDU

The Card SHALL respond to the *request\_pids()* APDU command by sending the *request\_pids\_cnf()* APDU when the Host is filtering PIDs with the PIDs it requires. The Card MAY send this APDU at any time. This list is absolute and SHALL always include all PIDs that the Card requires for a given transport stream.

**Table 9.12–10 - request\_pids\_cnf APDU**

Syntax	# of bits	Mnemonic
request_pids_cnf() { request_pids_cnf_tag length_field() ltsid number_of_pids for (i=0; i<number_of_pids; i++) { zero pid } }	24  8 8 3 13	uimsbf  uimsbf uimsbf uimsbf uimsbf

**request\_pids\_cnf\_tag** Value = 0x9FA017

**ltsid** Local Transport Stream ID. Only required when the M-CARD is present, and operating in M-Mode.

**number\_of\_pids** Number of PIDs required, maximum is 8

**zero** 3 bits of zero

**pid** PID value

## 9.13 Copy Protection

A Copy Protection Resource SHALL be opened as defined in Table 9.13–1 and SHALL comply to the interface requirement specifications as defined in the OC-SP-CCCP2.0 document [CCCP2].

**Table 9.13–1 - CableCARD Copy Protection Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Copy Protection	S-Mode	176	3	1	0x00B000C1
Copy Protection	M-Mode	176	4	1	0x00B00101

## 9.14 Extended Channel Support

For purposes of the Extended Channel, the Card or the Host that provides the physical communications link to the headend is referred to as the “link device”. The Card is the link device for the QPSK modem, and the Host is the link device for the Embedded Cable Modem (eCM).

The Extended Channel Support resource SHALL be created to register the applications that expect to send and receive data to and from the Extended Channel.

All Hosts are required to provide the hardware necessary to support a QPSK downstream out-of-band (OOB) channel for the Card, or to support the DOCSIS Set-top Gateway (DSG) function. The Card SHALL forward data received on this channel to the Host as appropriate through one or more data flows requested by the Host. In some cases, the Card will terminate data received on the QPSK downstream OOB channel or the DSG Ethernet tunnel for its own use (e.g., EMMs). In other cases, it MAY perform a filtering function and discard data known to be of no interest to the Host.

Supported system architectures imply two different ways of using the Extended Channel Support resource:

- The application is in the Host and the data are transferred to/from the headend via the QPSK modem.
- The application is in the Card and the data are transferred to/from the headend via the Host’s eCM.

This resource has four versions: Version 1 of this resource is required for Hosts that do not have an embedded High Speed Host (DOCSIS) Modem; Versions 2, 3, and 4 are required for Hosts that have an embedded High Speed Host (DOCSIS) Modem and support both Basic and Advanced DOCSIS Set-top Gateway (DSG). Version 4 of this resource includes support for all of the objects defined by versions 3, 2, and 1.

**Table 9.14–1 - Extended Channel Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Extended Channel Support	S-Mode/M-Mode	160	1	1	0x00A00041
Extended Channel Support	S-Mode/M-Mode	160	1	2	0x00A00042
Extended Channel Support	S-Mode/M-Mode	160	1	3	0x00A00043
Extended Channel Support	S-Mode/M-Mode	160	1	4	0x00A00044

The APDU messages are as follows:

**Table 9.14–2 - Extended Channel Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ Card	
			Host modem	Card modem
new_flow_req()	0x9F8E00	Extended Channel Support	↔	→
new_flow_cnf()	0x9F8E01	Extended Channel Support	↔	←
delete_flow_req()	0x9F8E02	Extended Channel Support	↔	→
delete_flow_cnf()	0x9F8E03	Extended Channel Support	↔	←
lost_flow_ind()	0x9F8E04	Extended Channel Support	↔	←
lost_flow_cnf()	0x9F8E05	Extended Channel Support	↔	→
inquire_DSG_mode()	0x9F8E06	Extended Channel Support	→	→
set_DSG_mode()	0x9F8E07	Extended Channel Support	←	←
DSG_error()	0x9F8E08	Extended Channel Support	←	N/A
DSG_message()	0x9F8E09	Extended Channel Support	→	N/A
configure_advanced_dsg()	0x9F8E0A	Extended Channel Support	←	N/A
send_DCD_info()	0x9F8E0B	Extended Channel Support	→	N/A

#### 9.14.1 new\_flow\_req APDU

The application SHALL use the *new\_flow\_req()* APDU to register a new flow with the link device.

For the case of when the DSG mode is selected, even though the Host has the modem, the Card SHALL request to open a new flow request indicating DSG as the service\_type. The Host will respond the new flow request and then the Host SHALL open an MPEG section flow to the Card for reception of the SI data.

The service types available are MPEG section, IP unicast, IP multicast, and DSG.

The Card and Host are required to support only one outstanding *new\_flow\_req()* APDU at a time. The Card SHALL send a *new\_flow\_cnf()* APDU with a status field of 0x04 (Network Busy) when additional *new\_flow\_req()* APDUs are received and one is pending.

**MPEG section** – This service type is applicable only for flows between the Card and the Host. The requested flow SHALL be in the form of MPEG table sections (both long and short form). This type of flow is unidirectional, from Card to Host only. The value of the section length field in these sections SHALL NOT exceed 4,093 bytes.

When the table section is in long form (as indicated by the section\_syntax\_indicator flag set to “1”), a 32-bit CRC is present. The 32-bit CRC is also present in short-form sections (as indicated by the section syntax indicator flag set to “0”) carried in the SI\_base\_PID (0x1FFC). For these table sections in which an MPEG-2 CRC is known to be present, the Card SHALL verify the integrity of the table section using the 32-bit CRC at the table section level, or a 32-bit CRC at another protocol layer. Only messages that pass the CRC check SHALL be forwarded to the Host. The Card SHALL discard table sections that are incomplete or fail the CRC check.

The 32-bit CRC MAY or MAY NOT be present in short-form sections associated with PID values other than the SI\_base\_PID (0x1FFC) and the Card MAY send these sections to the Host without any checks. In this case, the Host is responsible for validation of these sections.

**IP Unicast** – This service type applies both for flows between the Card and an eCM in the Host (DSG mode), and between the Host and an SCTE 55 modem in the Card (SCTE 55 mode). The requested flow SHALL be in the form

of IP packets addressed to or from the Card's IP address when in DSG mode, and the Host's IP address when in SCTE 55 mode. This type of flow MAY be bidirectional. The maximum total length of any IP packet in SCTE 55 mode SHALL be 1,500 bytes. With respect to DSG mode, the requested flow from the Card to the Host SHALL be in the form of IP packets addressed to the applicable destination IP address as determined by the Card. The requested flow from the Host to the Card in DSG mode SHALL be in the form of IP packets addressed to the IP address of the Card. The maximum total length of any IP packet in DSG mode SHALL be the DOCSIS maximum transmission unit (MTU), which is 1500 bytes. This MTU is relayed to the Card via the *new\_flow\_conf()* APDU.

**IP Multicast** – This service type is applicable both for flows between the Card and a modem in the Host, and for the Host and a modem in the Card. The requested flow SHALL be in the form of multicast IP packets addressed to the *multicast\_group\_ID* assigned IP address. This type of flow is unidirectional, from network to application only. The maximum total length of any IP packet SHALL be 1,500 bytes.

**DSG** – This service type is only applicable between the Host and the Card.

Conformance to this specification requires the Host and the Card to comply with the following requirements:

- The CHI SHALL support at least six concurrent MPEG section *service\_type* flows.
- The CHI SHALL support at least one IP unicast *service\_type* flow.
- The CHI SHALL support one DSG *service\_type* flow.
- The CHI SHALL support both TCP and UDP *flow\_types*.
- If the Service Information Virtual Channel Table indicates that one or more services are defined as being transported out-of-band, the Card SHALL provide one or more additional flows of the MPEG section *service\_type*.
- The Host SHALL support one DSG *Service\_type* flow to the Card, if requested, when a eCM is available.
- When the Host supports a unicast IP flow, it SHALL use DHCP per [RFC2131] to obtain an IP address for Card use. The Host SHALL provide the options parameters supplied by the Card in the *new\_flow\_req()* APDU to build the DHCP message, and add any other options as necessary or desired.
- The Card and Host are required to support only one outstanding *new\_flow\_req()* transaction at a time. The Card or Host SHALL send a *new\_flow\_cnf()* with a *Status\_field* of 0x04 (Network Busy) when additional *new\_flow\_req()* messages are received and one is pending.

**Table 9.14–3 - new\_flow\_req APDU Syntax**

Syntax	No. of Bits	Mnemonic
new_flow_req() {		
new_flow_req_tag	24	uimsbf
length_field()		
service_type	8	uimsbf
if (service_type == 00) { /* MPEG section */		
Reserved	3	bslbf
PID	13	uimsbf
}		
if (service_type == 01) { /* IP unicast */		
MAC_address	48	uimsbf
option_field_length	8	uimsbf
for (i=0; i<option_field_length; i++) {		
option_byte	8	uimsbf
}		
}		
if (service_type == 02) { /* IP multicast */		
Reserved	4	bslbf
multicast_group_ID	28	uimsbf
}		
}		

**new\_flow\_req\_tag** 0x9F8E00

**service\_type** Defines the type of requested service.

0x00 MPEG section  
0x01 IP unicast (IP\_U)  
0x02 IP multicast (IP\_M)  
0x03 DSG  
0x04-0xFF Reserved

**PID** The 13-bit MPEG-2 Packet Identifier associated with the flow request. The Card SHALL be responsible for filtering the MPEG-2 transport stream and delivering only MPEG table sections delivered on transport packets with the given value of PID.

**MAC\_address** The 48-bit MAC address of the entity requesting the unicast IP flow.

**option\_field\_length** The number of bytes in the following for loop.

**option\_byte** These bytes correspond to the options field of a DHCP message. One or more DHCP options per [RFC2132] MAY be included. The “end option” (code 255) SHALL NOT be used, so that the entity granting the IP flow request MAY append zero or more additional option fields before delivering the request to the server.

**multicast\_group\_ID** The multicast group ID associated with the flow request. The modem function SHALL be responsible for filtering arriving multicast IP packets and delivering only packets matching the given multicast\_group\_ID address.

#### 9.14.1.1 new\_flow\_req IP Unicast DSG Mode Details

When the Host is configured for DSG mode, then the Host and Card SHALL interact as defined within this section:

- If the Card requires two-way communications in DSG mode, then the Card SHALL request a new IP Unicast flow using the new\_flow\_request() APDU

- When requesting the new flow, the Card SHALL at a minimum supply the Vendor Specific Options defined in Table 9.14–6 and provide the MAC address of the Card in the MAC\_address field.
- If the Host grants the request for the IP flow, then the Host SHALL acquire an IP address for the CARD and forward this information to the Card in the IP\_address field of the new\_flow\_cnf() APDU, the maximum transmission unit (MTU) for the DOCSIS network (1500 bytes for IP datagrams) in the max\_pdu\_size field and any requested options via the option\_byte field.
- The Host SHALL acquire a unique IP address for the Card utilizing DHCP as defined in [RFC2131].
- The Host SHALL be responsible for resolving Card IP address conflicts as defined in [RFC2131].
- All Host DHCP transactions associated with acquiring the Card and IP address SHALL be over the eCM interface and SHALL not propagate to any other interface on the Host.
- The Host SHALL not use Network Address Translation (NAT) to provide an IP address for the Card.
- On receipt of an IP packet from the Card over the interface via the IP Unicast Flow, the Host SHALL acquire the appropriate destination MAC address, encapsulate the IP packet received over the interface within Ethernet frames utilizing the acquired destination MAC address.
- On receipt of an Ethernet frame from the eCM interface targeted to the MAC address of the Card, the Host SHALL extract the embedded IP packet and forward the packet to the Card via the granted IP Unicast Flow.
- The Host SHALL not forward any Ethernet frames or IP packets destined to the Card to any interface other than the CHI.
- The Host SHALL only forward IP packets destined to the Card that have been received via the eCM interface or via applications resident on the Host.
- The Host SHALL NOT forward any IP packets received from the Card over the CHI to any interface other than the eCM interface.
- The CARD SHALL implement the DHCP Vendor Specific Information Option (option 43) and Vendor Class Identifier Option (option 60) as specified in Table 9.14–4 and Table 9.14–5.

**Table 9.14–4 - Card DHCP Vendor Specific Information (Option 43) Sub-option Encoding**

Sub-option	Value	Description
1	"<null>"	The request sub-option vector is a list of sub-options (within option 43) to be returned to client by the server upon reply to the request. None defined.
2	"CARD"	Device type of the entity making the DHCP request.
3	"ECM:ESTB:CARD"	Indicates that a CARD is making a request via the eCM's DOCSIS return channel
4	"<device serial number>"	Serial Number of CARD. If Serial Number is not available, then other unique identifier (other than MAC Address) may be utilized
5	"<hardware version number>"	Hardware version number of CARD
6	"<firmware version number>"	Firmware version number of CARD
7	"<boot ROM version number>"	Boot ROM version number of CARD

Sub-option	Value	Description
8	e.g., “0204DF”	A 6-octet, hexadecimal-encoded, vendor-specific Organization Unique Identifier (OUI) that may match the OUI in the eCM’s MAC address.
9	e.g., “XYZ-CARD-001”	Vendor model number of CARD
51	e.g., “XYZ Corporation”	Vendor name
52	“yyyyyy”	CARD capability using the encoding format per DOCSIS specification. Since there is no standard/required capability identification, Conditional Access vendor must provide documentation on the supported capability.
53	e.g., “000-01234-56789-000” (example is unit address of Motorola CARD)	Conditional Access Vendor specific device identification
54	e.g., “00AA11BB22CC33DD”	64 bit CARD_ID as specified in CARD X.509 certificate

**Table 9.14–5 - Card DHCP Vendor Class Identifier (Option 60) Encoding**

Option	Value	Description
60	“OpenCable 2.0”	OpenCable Version

### 9.14.2 new\_flow\_cnf APDU

The link device SHALL return a *new\_flow\_cnf()* APDU after receiving a *new\_flow\_req()* APDU.

The Card is required to support only one outstanding new\_flow\_req transaction at a time. The Card SHALL send a *new\_flow\_cnf()* with a status field of 0x04 (Network Busy) when additional new\_flow\_req messages are received and one is pending.



**Table 9.14–6 - new\_flow\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
new_flow_cnf() {		
new_flow_cnf_tag	24	uimsbf
length_field()		
status_field	8	uimsbf
flows_remaining	8	uimsbf
if (status_field == 0x00) {		
flow_id	24	uimsbf
service_type	8	uimsbf
if (service_type == IP_U) {		
IP_address	32	uimsbf
flow_type	8	uimsbf
flags	3	uimsbf
max_pdu_size	13	uimsbf
option_field_length	8	uimsbf
for (i=0; i<option_field_length; i++) {		
option_byte	8	uimsbf
}		
}		
}		
}		

**new\_flow\_cnf\_tag**

0x9F8E01

**status\_field**

Returns the status of the new\_flow\_req.

- 0x00 Request granted, new flow created
- 0x01 Request denied, number of flows exceeded
- 0x02 Request denied, service\_type not available
- 0x03 Request denied, network unavailable or not responding
- 0x04 Request denied, network busy
- 0x05 Request Denied – MAC address not accepted
- 0x06-0xFF Reserved

**flows\_remaining**

The number of additional flows of the same service\_type that can be supported. The value 0x00 indicates that no additional flows beyond the one currently requested can be supported.

**flow\_id**

The unique flow identifier for this application's data flow. The flow\_id value of 0x000000 is reserved and SHALL NOT be assigned.

**service\_type**The requested service\_type received in the *new\_flow\_req()* APDU.**IP\_address**

The 32-bit IP address associated with the requested flow.

**flow\_type**

An 8-bit unsigned integer number that represents the protocol(s) supported by the Card to establish the IP-U flow. The field has the following values:

- 0x00 UDP and TCP supported
- 0x01 UDP only supported
- 0x02 TCP only supported
- 0x03-FF Reserved

**flags**

A 3-bit field that contains information, as defined below, pertaining to limitations associated with the interactive network. Additional detail is provided in Table 9.14–7.

- Bit 0 no\_frag
- bits 2:1 reserved

**Table 9.14–7 - Flag field definitions**

BITS		
2	1	0
reserved		no_frag

<b>no_frag</b>	A 1-bit Boolean that designates if the network supports fragmentation. A value of 0 <sub>2</sub> indicates that fragmentation is supported. A value of 1 <sub>2</sub> indicated that fragmentation is not supported.
<b>max_pdu_size</b>	A 13-bit unsigned integer number that designates the maximum PDU length that MAY be transmitted across the interface.
<b>option_field_length</b>	An 8-bit unsigned integer number that represents the number of bytes of option field data to follow.
<b>option_byte</b>	These bytes correspond to the options requested in the <i>new_flow_req()</i> message. The format of the field is as defined in [RFC2132]. The end option (code 255) SHALL NOT be used.

### 9.14.3 delete\_flow\_req APDU

The application SHALL use the *delete\_flow\_req()* APDU to delete a registered data flow.

**Table 9.14–8 - delete\_flow\_req APDU Syntax**

Syntax	No. of Bits	Mnemonic
delete_flow_req() { delete_flow_req_tag length_field() flow_id }	24	uimsbf
	24	uimsbf

**delete\_flow\_req\_tag** 0x9F8E02

**flow\_id** The flow identifier for the flow to be deleted.

### 9.14.4 delete\_flow\_cnf APDU

When the link device receives a *delete\_flow\_req()* APDU, it SHALL respond with the *delete\_flow\_cnf()* APDU.

**Table 9.14–9 - delete\_flow\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
delete_flow_cnf() { delete_flow_cnf_tag length_field() flow_id status_field }	24	uimsbf
	24	uimsbf
	8	uimsbf

**delete\_flow\_cnf\_tag** 0x9F8E03

**flow\_id** The flow identifier for the flow to be deleted.

**status\_field** Returns the status of the *delete\_flow\_req()* APDU.  
0x00 Request granted, flow deleted

- 0x01 Reserved
- 0x02 Reserved
- 0x03 Request denied, network unavailable or not responding
- 0x04 Request denied, network busy
- 0x05 Request denied, flow\_id does not exist
- 0x06 Request denied, not authorized
- 0x07-0xFF Reserved

#### 9.14.5 lost\_flow\_ind APDU

A link device SHALL indicate that a registered data flow has been lost by issuing the *lost\_flow\_ind()* APDU.

**Table 9.14–10 - lost\_flow\_ind APDU Syntax**

Syntax	No. of Bits	Mnemonic
lost_flow_ind() {		
lost_flow_ind_tag	24	uimsbf
length_field()		
flow_id	24	uimsbf
reason_field	8	uimsbf
}		

**lost\_flow\_ind\_tag** 0x9F8E04

**flow\_id** The flow identifier for the flow that has been lost.

**reason\_field** Returns the reason the flow was lost.

- 0x00 Unknown or unspecified reason
- 0x01 IP address expiration
- 0x02 Network down or busy
- 0x03 Lost or revoked authorization
- 0x04-0xFF Reserved

#### 9.14.6 lost\_flow\_cnf APDU

The application SHALL respond with the *lost\_flow\_cnf()* APDU when a *lost\_flow\_ind()* APDU is received.

**Table 9.14–11 - lost\_flow\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
lost_flow_cnf() {		
lost_flow_cnf_tag	24	uimsbf
length_field()		
flow_id	24	uimsbf
status_field	8	uimsbf
}		

**lost\_flow\_cnf\_tag** 0x9F8E05

**flow\_id** The flow identifier for the flow that has been lost.

**status\_field** Returns the status of the *lost\_flow\_ind()* APDU.

- 0x00 Indication acknowledged
- 0x01-0xFF Reserved

#### 9.14.7 DSG Mode

There are two different operational modes defined for DSG, Basic and Advanced Mode. Of these two modes, there is also two different states for each of these modes: DSG Mode, indicating that a RDC is present and has the ability

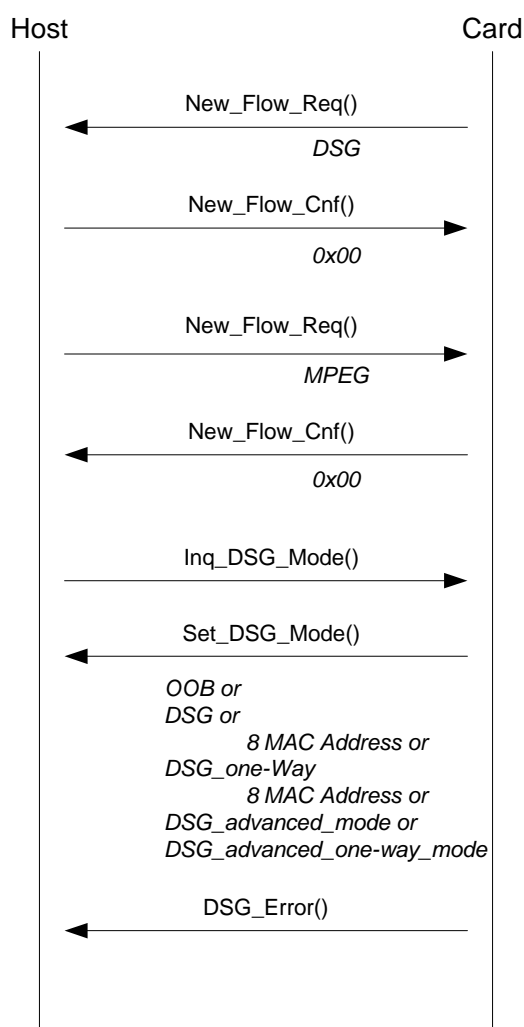
to communicate back to the headend; and DSG-One-Way\_mode, where the RDC is not present, or is not active, and there is no communication back to the headend. The DSG\_Mode is the desired “Normal” Operating mode.

For DSG Basic Mode Operation:

- The Card SHALL provide the Host with a set of MAC Addresses that the eCM SHALL use to filter DSG tunnels.
- The eCM SHALL utilize the presence/absence of the requested tunnel MAC Address to determine if a downstream channel contains valid DSG tunnels.
- The Host SHALL NOT forward the DCD messages, if present, to the Card.

Setting this mode is equivalent to the state Notification from DSG Client Controller: enable upstream transmitter defined in the DSG specification.

The following figure is an example of the initial message exchange between the Card and the Host for DSG Basic Mode operation:



**Figure 9.14-1 - DSG Mode Message Flow**

#### 9.14.7.1 *inquire\_DSG\_mode* APDU

The Host SHALL use the *inquire\_DSG\_mode* () object to inquire the preferred operational mode for the network.

The Host SHALL inquire from the Card the preferred operational mode for the network, either OOB mode or DSG mode by sending the *inquire\_DSG\_mode()* APDU.

**Table 9.14–12 - inquire\_DSG\_mode APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>inquire_DSG_mode() {     inquire_DSG_mode_tag     length_field() }</pre>	24	uimsbf

**inquire\_DSG\_mode\_tag**            0x9F8E06

#### 9.14.7.2 set\_DSG\_mode APDU

The Card SHALL use the *set\_DSG\_mode()* APDU to inform the Host of the preferred operational mode for the network. This message is sent in response to the *inquire\_DSG\_mode()*, or it MAY be sent as an unsolicited message to the Host after the resource session has been established. The method by which the Card determines the preferred operational mode is proprietary to the CA/Card system vendor. The *set\_DSG\_mode()* SHALL be used to indicate either OOB\_Mode or DSG\_mode, DSG\_One-Way\_Mode, advanced\_DSG\_mode or advanced\_DSG\_one-way\_mode.

A default operational mode SHALL be utilized when the Host and/or Card is unable to obtain the preferred operational mode. There are two potential default conditions that SHALL be addressed.

- Either the Host or the Card MAY NOT support version 2 of the Extended Channel Support resource (*inquire\_DSG\_mode* and *set\_DSG\_mode()* APDUs).
- The Card MAY NOT have acquired the preferred operational mode from the network due to possible network errors.

To ensure backward compatibility in the first case above, a Host SHALL initialize in the default operational mode of OOB\_mode. In the second case, the Card SHOULD instruct the Host that the preferred operational mode is OOB\_mode.

If the operational mode is DSG\_mode, DSG\_one-way\_mode, advanced\_dsg\_mode or advanced\_dsg\_one-way\_mode, the Card SHALL provide up to eight Ethernet MAC addresses and number of header bytes to be removed from the DSG tunnel packets. In DSG or DSG\_one-way mode, once the DSG extended channel flow has been opened, the Host SHALL filter IP packets whose Ethernet destination address match any of the specified DSG\_MAC\_address values, remove the specified number of header bytes from these packets, before sending these packets across the extended channel.

**Table 9.14–13 - set\_DSG\_mode APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>set_DSG_mode() {     set_DSG_mode_tag     length_field()     operational_mode     if ((operation_mode == DSG_mode)            (operation_mode == DSG_one-way_mode)) {         number_MAC_addresses         for (i=0; i&lt;number_MAC_addresses; i++) {             DSG_MAC_address         }         remove_header_bytes     } }</pre>	24	uimsbf
	8	uimsbf
	8	uimsbf
	48	uimsbf
	16	uimsbf

**set\_DSG\_mode\_tag**

0x9F8E07

**operational\_mode**

Defines the preferred operational mode of the network.

0x00 **OOB\_mode** – In this mode, the reverse OOB transmitter is under control of the Card through the use of the **OOB\_TX\_tune\_req()** APDU in the Host Control resource. The Host SHALL respond to these messages by tuning the reverse OOB transmitter to the requested frequency and coding value (bit-rate and power level). The Card uses the OOB-RDC for returning data to the cable headend.

0x01 **DSG\_mode** – In this mode the Host uses the eCM as the transmitter for the reverse path. If the Card attempts to command the reverse OOB transmitter with the **OOB\_TX\_tune\_req()** APDU while the Host is operating in DSG mode, the Host will deny the tune request with a “Tuning Denied – RF Transmitter Busy” status. Also, in this mode, the receiver for the OOB FDC is not active. If the Card attempts to command this receiver with the **OOB\_RX\_tune\_req()** message while the Host is operating in the DSG mode, the Host SHALL deny the tune request with a “Tuning Denied – Other reasons” status.

0x02 **DSG\_one-way\_mode** – In this mode, the reverse OOB transmitter and eCM transmitter SHALL be disabled for both the RDC and the DOCSIS return channel. Also, in this mode, the receiver for the OOB FDC is not active. If the Card attempts to command this receiver with the **OOB\_RX\_tune\_req()** message while the Host is operating in the DSG one-way mode, the Host SHALL deny the tune request with a “Tuning Denied – Other reasons” status. If the Card attempts to command the reverse OOB transmitter with the **OOB\_TX\_tune\_req()** APDU while the Host is operating in DSG mode, the Host will deny the tune request with a “Tuning Denied – Other Reasons”. This mode could be used in one-way cable systems and for network diagnosis in two-way cable systems.

0x03 **advanced\_dsg\_mode** – In this mode, the Host uses the eCM as the transmitter for the reverse path. If the Card attempts to command the reverse OOB transmitter with the **OOB\_TX\_tune\_req()** message while the Host is operating in the DSG mode, the Host SHALL deny the tune request with a “Tuning Denied – RF Transmitter busy” status. Also, in this mode, the receiver for the OOB FDC is not active. If the Card attempts to command this receiver with the **OOB\_RX\_tune\_req()** message while the Host is operating in the DSG mode, the Host SHALL deny the tune request with a “Tuning Denied – Other reasons” status. Setting this mode is equivalent to the state Notification from DSG Client Controller: enable upstream transmitter defined in the DSG specification.

0x04 **advanced\_dsg\_one-way\_mode** – In this mode, the reverse OOB transmitter and eCM Transmitter SHALL be disabled for both the RDC and the DOCSIS return channel. Also, in this mode, the receiver for the OOB FDC is not active. If the Card attempts to command this receiver with the **OOB\_RX\_tune\_req()** message while the Host is operating in the DSG one-way mode, the Host SHALL deny the tune request with a “Tuning Denied – Other reasons” status. If the Card attempts to command the reverse OOB transmitter with the **OOB\_TX\_tune\_req()** APDU while the Host is operating in DSG mode, the Host will deny the tune request with a “Tuning Denied – Other Reasons”. This mode could be used for network diagnosis in two-way cable systems. Setting this mode is equivalent to the state Notification from DSG Client Controller: disable upstream transmitter defined in the DSG specification.

NOTE: Operating the Host in this mode will interrupt all two-way IP connectivity until another mode is selected.

	05-0xFF Reserved
<b>number_MAC_addresses</b>	The number of DSG MAC addresses allocated by the Card provider to carry DSG tunnels. A maximum of eight DSG tunnels per Card provider are allowed.
<b>DSG_MAC_address</b>	The Ethernet MAC addresses allocated by the Card provider to carry the DSG tunnels.
<b>remove_header_bytes</b>	The number of bytes to be removed from the DSG tunnel packets before delivery over the extended channel. A value of zero implies that no header bytes are to be removed.

For the DSG Advanced Mode:

- The Host SHALL scan downstream channels for DCD messages upon receipt of a *set\_dsg\_mode* () object with a value = 0x03 or 0x04.
- The Host SHALL pass the received DCDs to the Card using the *send\_DCD\_info* () object only when the Host detects a change in the configuration count change field in the DCD message. The DCD message is defined in [DSG].
- The Card SHALL determine if the DCD tunnel addresses are valid and inform the Host if the DSG channel is not valid.
- The Card utilizes the **DSG\_error()** APDU to indicate that the DCD message is not valid.
- If the DSG channel is not valid, e.g., no CA Tunnel present, then the Host SHALL search a new downstream channel for a DCD message.
- If the DSG channel is valid, then the Host SHALL stay on the downstream and forward requested tunnels to the Card.
- Upon selection of a valid downstream, the Card SHALL pass the DSG Configuration information received in the DCD to the Host using *configure\_advanced\_DSG* ().
- The Host SHALL use the *dsg\_message* () to pass the UCID, when identified, to the Card.
- The Card SHALL be capable of using the Upstream Channel ID (UCID) passed by the Host in the *dsg\_message* () to select appropriate tunnels when UCIDs are specified in the DSG rules.
- The Host SHALL use *dsg\_message* () to pass application\_id(s) to the Card.
- After parsing the DCD message for desired DSG tunnels, the Card uses the *configure\_advanced\_DSG* () object to provide the Host with a set of MAC Addresses and DSG classifiers as applicable, that the eCM SHALL use to filter DSG Tunnels.
- Host specific tunnels are indicated by the presence of the requested application ID, that is, the application ID does not equal zero (0).
- DSG Tunnels addresses with an application ID of (0) are requested by the Card.
- The Card SHALL not request any tunnels with a UCID other than the UCID passed by the Host in the *dsg\_message* ().

The following figure is an example of the initial message exchange between the Card and the Host for Advanced Mode Operation:

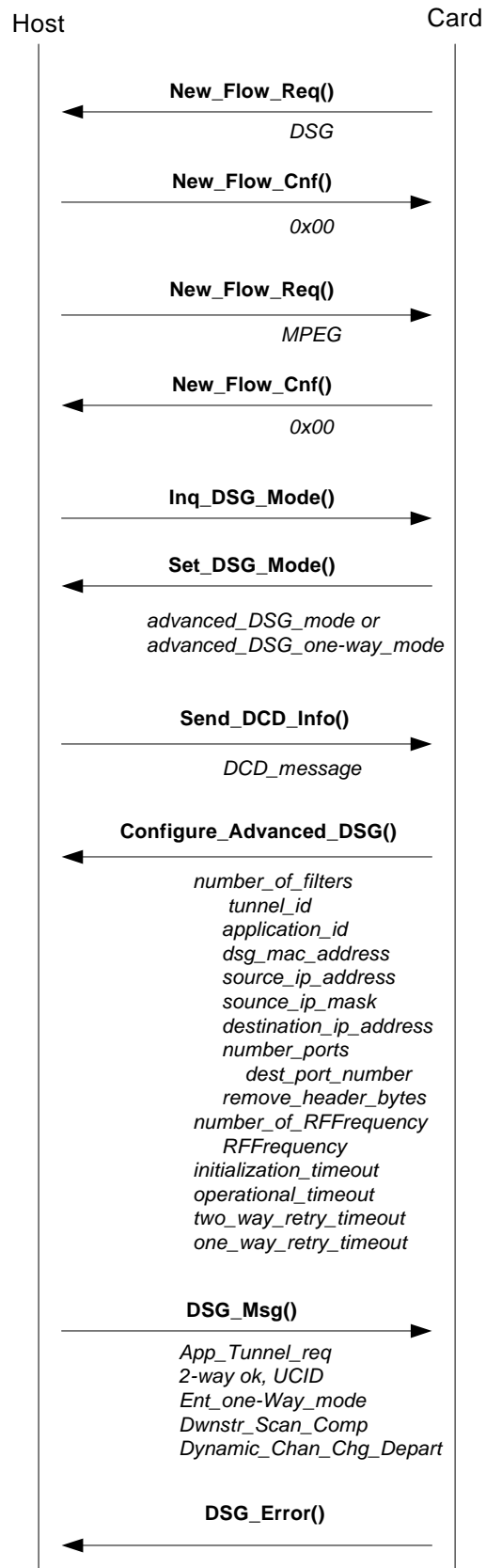


Figure 9.14-2 - Sample Advanced Mode Message Flow



### 9.14.7.3 *configure\_advanced\_DSG* APDU

The Card SHALL use the *configure\_advanced\_DSG()* object to pass DSG Advanced Mode configuration parameters to the eCM if the Card supports DSG Advanced mode and the Host reports an extended channel resource version set to 3 or 4. This message is sent in response to the *send\_DCD\_info()* APDU message.

MAC Addresses and DSG classifiers provided in the *configure\_advanced\_DSG()* object override all previously defined values passed by the Card.

When the operational mode is either *Advanced\_DSG\_mode* or *Advanced\_DSG\_One-Way\_mode*, then the Card may provide up to eight unique Ethernet MAC addresses along with a set of DSG classifiers. The Card may also specify the number of header bytes to be removed from the DSG tunnel packets.

In *Advanced\_DSG* or *Advanced\_DSG\_One-Way* mode, the eCM/Host SHALL forward IP packets whose MAC destination address and layer-3/layer-4 parameters match any of the combinations of DSG\_MAC address and DSG\_classifiers specified in the *configure\_advanced\_DSG()* object.

- When an IP Packet matches a DSG MAC Address/DSG Classifier combination, that packet SHALL be forwarded.
- If a DSG classifier is not provided for a specific DSG MAC address, the Host SHALL forward all Ethernet frames received on that MAC address.

The *Application\_ID* parameter is used by the Card to signal the intended destination for the packets for each DSG MAC Address/DSG Classifier combination.

- An *Application\_ID* of zero (0) indicates that the Host SHALL forward matching packets to the Card.
- An *Application ID* greater than zero (0) indicates that the matching packets SHALL terminate at the eCM/Host.

The Host SHALL remove the specified number of header bytes from these packets before delivery across the extended channel interface to the Card.

**Table 9.14–14 - Configure Advanced DSG Object Syntax**

Syntax	# of Bits	Mnemonic
<i>configure_advanced_DSG</i> () {		
<i>configure_advanced_DSG_tag</i>	24	uimsbf
<i>length_field()</i>		
<i>number_of_filters</i>	8	uimsbf
for (i=0; i< <i>number_tunnel_filters</i> ; i++) {		
<i>tunnel_id</i>	8	uimsbf
<i>application_id</i>	16	uimsbf
<i>dsg_mac_address</i>	48	uimsbf
<i>source_IP_address</i>	32	uimsbf
<i>source_IP_mask</i>	32	uimsbf
<i>destination_IP_address</i>	32	uimsbf
<i>number_ports</i>	8	uimsbf
for(i=0; i< <i>number_ports</i> ; i++){		
<i>dest_port_number</i>	16	uimsbf
}		
<i>remove_header_bytes</i>	16	uimsbf
}		
<i>number_of_RXFrequency</i>	8	uimsbf
for (i=0; i< <i>number_of_RXFrequency</i> ; i++){		
<i>RXFrequency</i>	32	uimsbf
}		
<i>initialization_timeout</i>	16	uimsbf

Syntax	# of Bits	Mnemonic
operational_timeout	16	uimsbf
two_way_retry_timeout	16	uimsbf
one_way_retry_timeout	16	uimsbf
}		

**configure\_advanced\_DSG\_tag** 0x9F8E0A

**number\_of\_filters** The number of DSG tunnels that the Host eCM SHALL filter. A maximum of eight unique tunnel filters are allowed, although this number may be greater than eight if certain MAC Addresses are used by multiple DSG tunnels.

**tunnel\_id** An Identifier for the tunnel. This field should match the DSG Rule ID received in the DCD message for the tunnel identifier. The tunnel\_id is used by the eCM to populate the `dsgIfStdTunnelFilterTunnelId` MIB object.

**application\_id** The application ID associated with the requested DSG tunnel. A value of zero (0) indicates that the DSG tunnel is requested by the Card and SHALL be passed to the Card. A value other than zero (0) indicates that the tunnel is requested by the Host, which SHALL be terminated in the Host, and is the same value that was passed to the Card by the Host in the *dsg\_message()* object.

**dsg\_mac\_address** The MAC addresses to be filtered by the eCM.

**source\_IP\_address** The IP source address specified in the DCD message to be used in layer 3 filtering. A value of all zeros implies all values of Source IP Address, i.e., this parameter was not specified in the DCD message.

**source\_IP\_mask** The source IP mask specified in the DCD message to be used in layer 3 filtering. A value of all ones implies that all 32 bits of the Source IP Address are to be used for filtering.

**destination\_IP\_address** The IP destination address specified in the DCD message to be used in layer 3 filtering. A value of all zeros implies all values of Destination IP Address, i.e., this parameter was not specified in the DCD message.

**number\_ports** The number of TCP/UDP Destination Port numbers associated with a DSG\_MAC\_Address.

**dest\_port\_number** The range of TCP/UDP Destination Port addresses specified in the DCD message, listed here as individual port numbers.

**remove\_header\_bytes** The number of bytes to be removed from the DSG tunnel packets before delivery. A value of zero implies that no header bytes be removed.

**number\_of\_RXFrequency** The number of TLV channel list entry in the DCD message.

**RXFrequency** The RX Frequency as defined in [DSG]

**initialization\_timeout** DSG Initialization Timeout (Tdsg1). The timeout period for the DSG packets during initialization as defined in [DSG]. A value of zero indicates that the default value SHALL be used.

**operational\_timeout** DSG Operational Timeout (Tdsg2). The timeout period for the DSG packets during normal operation as defined in [DSG]. A value of zero indicates that the default value SHALL be used.

**two\_way\_retry\_timeout** DSG Two-Way Retry Timer (Tdsg3). The retry timer that determines when the DSG eCM attempts to reconnect with the CMTS as defined in [DSG]. A value of zero indicates that the default value SHALL be used.

**one\_way\_retry\_timeout** DSG One-Way Retry Timer (Tdsg4). The retry timer that determines when the DSG eCM attempts to rescan for a downstream DOCSIS channel that contains DSG packets as defined in [DSG]. A value of zero indicates that the default value SHALL be used.

#### 9.14.7.4 DSG\_message APDU

The Host SHALL use the *dsg\_message* () object to request Application tunnel data streams, to indicate that the eCM has established two-way communication and is passing the UCID of the upstream channel, to indicate that the eCM has entered One-way mode, to indicate that the eCM has done a complete downstream scan without finding a DCD message or a Basic Mode tunnel, or to indicate that the eCM has received a DCC-REQ message and is preparing to execute a Dynamic Channel Change.

**Table 9.14–15 - DSG Message Object Syntax**

Syntax	# of bits	Mnemonic
<code>dsg_message () {</code>		
<code>dsg_message_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>message_type</code>	8	uimsbf
<code>If (message_type = 0x00) {</code>		
<code>number_app_ids</code>	8	uimsbf
<code>for (i=0; i &lt; number_app_ids; i++) {</code>		
<code>application_id</code>	16	uimsbf
<code>}</code>		
<code>}</code>		
<code>If (message_type = 0x01) {</code>		
<code>UCID</code>	8	uimsbf
<code>}</code>		
<code>if (message_type = 0x04) {</code>		
<code>init_type</code>	8	uimsbf
<code>}</code>		
<code>}</code>		

**dsg\_message\_tag**

0x9F8E09

**message\_type**

Indicates the purpose of the object as defined below.

**0x00 Application\_tunnel\_request** — the Host has determined that there are applications that require data from one or more DSG tunnels. The Host passes the application\_id(s) of applications requesting access to DSG tunnels to the Card. The Card parses the DCD message and provides MAC Address and DSG classifiers for the requested application tunnels. This is only used in DSG Advanced mode.

**Number\_app\_ids** – the total number of application IDs to follow; only valid when message type is Application\_tunnel\_request.

**Application\_ID** – the application ID of the DSG Application tunnel required by the Host. The application\_ID MAY be obtained from the source\_name\_subtable of the Network Text Table contained in ANSI/SCTE 65. The Card utilizes the application ID to parse the DCD for the presence of the requested application tunnel. If the tunnel is present, then the Card uses the *configure\_advanced\_DSG()* object to pass the MAC Address and DSG classifiers associated with the requested application tunnel to the Host.

0x01 2-way OK, UCID – the Host has established two-way communication and is providing the Card with the channel ID (UCID) of the upstream channel.

**Advanced Mode:** The Card uses this value for filtering of various DSG rules as applicable.

**Basic Mode:** The Card SHALL ignore this value.

**UCID** – the channel ID of the DOCSIS channel that the Host is using for upstream communication.

0x02 Entering\_One-Way\_mode – Sent from the Host to the Card as an indicator that a timeout or other condition has forced the eCM into One-Way operation.

0x03 Downstream Scan Completed – Sent from the Host to the Card after a complete downstream scan as an indicator that the eCM,

**Advanced Mode:** Has been unable to identify a downstream channel with a DCD message.

**Basic Mode:** Has been unable to find a DSG tunnel with a well-known MAC address.

0x04 Dynamic Channel Change (Depart) – the eCM has transmitted a DCC-RSP (Depart) on the existing upstream channel and is preparing to switch to a new upstream or downstream channel. After channel switching is complete, the eCM transmits a DCC – RSP (Arrive) to the CMTS unless the MAC was reinitialized. In either case the eCM will resend DSG\_message() with message\_type 0x01 “2-way OK, UCID” to indicate the upstream has been established.

**Init\_type** – specifies what level of reinitialization the eCM will perform, if any, before communicating on the new channel(s), as directed by the CMTS.

0x00 = Reinitialize the MAC

0x01 = Perform broadcast initial ranging on new channel before normal operation

0x02 = Perform unicast initial ranging on new channel before normal operation

0x03 = Perform either broadcast initial ranging or unicast initial ranging on new channel before normal operation

0x04 = Use the new channel(s) directly without re-initializing or initial ranging

0x05 = Reinitialization method not specified

0x06 – 0xFF Reserved

#### 9.14.7.4.1 Dynamic Channel Change (Informative)

Dynamic Channel Change operations can cause a DSG eCM to move to a new upstream and/or downstream channel(s) either through manual intervention at the CMTS or autonomously via a load-balancing operation.

**Message\_type** = 0x01 and 0x04 allow the DSG Client Controller to be made aware of the initiation and progress of DCC operations. Acting upon these messages, the Client Controller can provide the proper reaction to upstream and downstream channel changes; in particular, the Client Controller should take action to make sure it still has a valid DSG channel after the DCC operation has completed.

#### 9.14.7.5 send\_DCD\_info APDU

The *send\_DCD\_info()* is an APDU used to pass DCD information between the Host and Card. In DSG Advanced mode, the Host SHALL use the *send\_DCD\_info()* object to pass the entire DCD message, except for the DOCSIS MAC Management header, to the Card. Upon receipt of the DCD message, the Card SHALL parse the DCD information.

**Table 9.14–16 - send\_DCD\_info Object Syntax (Resource Version 3)**

Syntax	# of bits	Mnemonic
<pre>send_DCD_info () {     send_DCD_info _tag     length_field()     DCD_message }</pre>	24  ( *)	uimsbf

**send\_DCD\_info\_tag** 0x9F8E0B

**DCD\_message** The TLVs comprising the DCD message as defined in [DSG] in the Summary of DCD TLV Parameters table.

#### 9.14.7.6 DSG\_error APDU

The Card MAY inform the Host of errors that occur in receiving DSG packets using the **DSG\_error()** APDU.

**Table 9.14–17 - DSG\_error APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>DSG_error() {     DSG_error_tag     length_field()     error_status }</pre>	24	uimsbf
	8	uimsbf

**DSG\_error\_tag** 0x9F8E08

**error\_status**

Indicates the type of error that occurred

0x00 Byte count error – The Card did not receive the same number of bytes in the DSG packet as was signaled by the Host.

0x01 Invalid\_DSG\_channel –

**Advanced Mode:** The Current DCD message transmitted to the Card is not valid or does not contain the requested DSG tunnel(s). The Host SHALL acquire a new DCD on a different downstream and pass this DCD to the Card. Sent from the Card to the Host during initial tunnel acquisition or when a DCD no longer contains a required tunnel.

**Basic Mode:** The current DSG channel is not valid. The Host SHALL find another DSG channel that contain DSG tunnels with the well-known MAC address(es).

0x02 Application\_ID\_error – The current DCD message transmitted to the Card does not contain a valid entry for an application ID requested by the Host. The Host MAY choose to not to wait for data intended for the specified application from that tunnel if Application\_ID is invalid.

0x03-0xFF Reserved

## 9.15 Generic Feature Control

The Generic Feature Control resource enables the Host device to receive control of features, which are considered generic to Host devices. There are three aims to this resource: 1) to provide control of features that subscribers do not desire to set themselves, 2) to provide the ability to inhibit subscriber control and only allow headend control, and 3) to provide a mechanism in which a Card or Host device can be staged to a known value.

A resource is created which resides in the Host called the Generic Feature Control resource. If the Host reports this resource to the Card, the Card SHALL open only one session to the Host and SHOULD never close the session.

### 9.15.1 Parameter Storage

#### 9.15.1.1 Host

The Host MAY provide non-volatile storage for the parameters associated with generic features on a parameter-by-parameter basis. These parameters SHALL be stored in the Host.

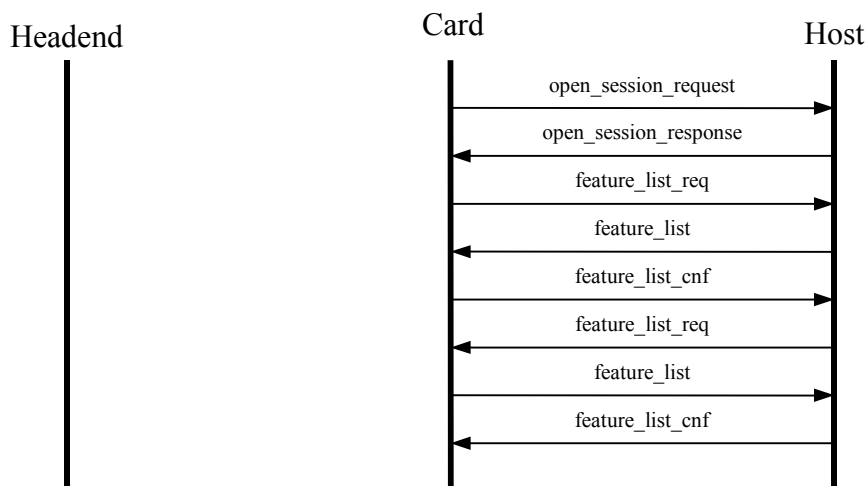
#### 9.15.1.2 CableCARD Device

There is no requirement for the Card to store the generic feature's parameters although there is no requirement that it cannot.

### 9.15.2 Parameter Operation

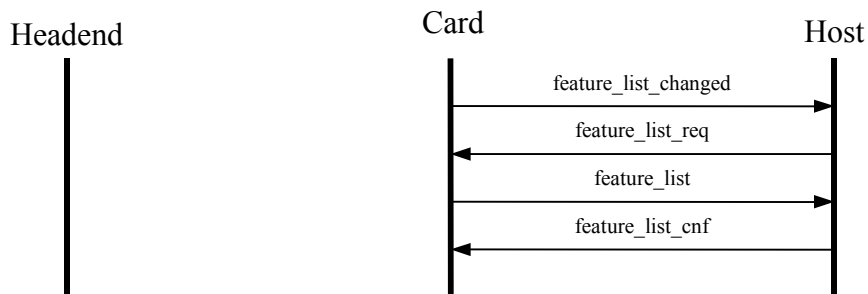
#### 9.15.2.1 Feature List Exchange

Immediately after the session to the Generic Feature Control resource has been established, the Card SHALL query the Host to determine which generic features are supported in the Host (*feature\_list\_req*). After the Card receives the generic feature list from the Host (*feature\_list*), the Card SHALL send its confirmation of the feature list to the Host (*feature\_list\_cnf*). The Host SHALL then query the Card to determine which generic features are supported in the Card and the headend (*feature\_list\_req*). The Card SHALL send its feature list to the Host (*feature\_list*) to which the Host SHALL send its confirmation (*feature\_list\_cnf*). This is called the generic feature list exchange.

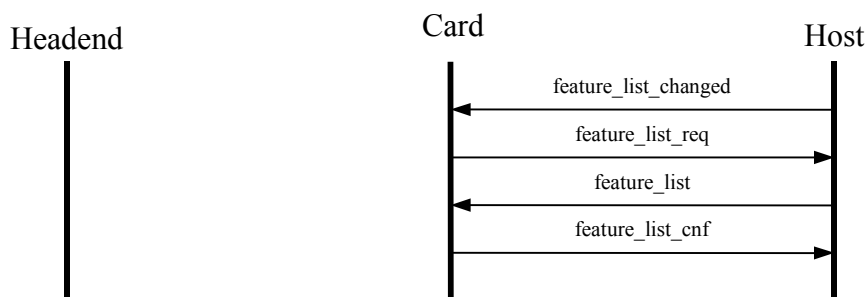


**Figure 9.15-1 - Generic Feature List Exchange**

If the generic feature list on the Host or the Card changes, then the changed device SHALL send a *feature\_list\_changed()* APDU to the other device. The other device SHALL then perform the generic feature list exchange to obtain the new list.



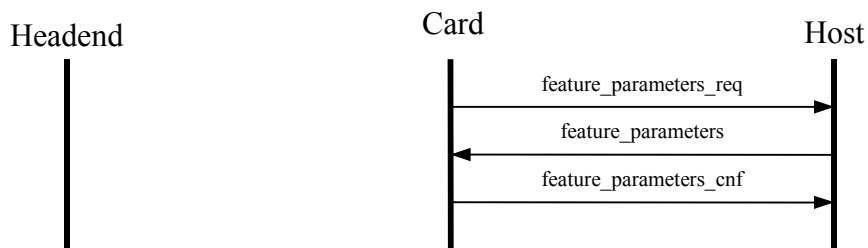
**Figure 9.15-2 - Card Feature List Change**



**Figure 9.15-3 - Host Feature List Change**

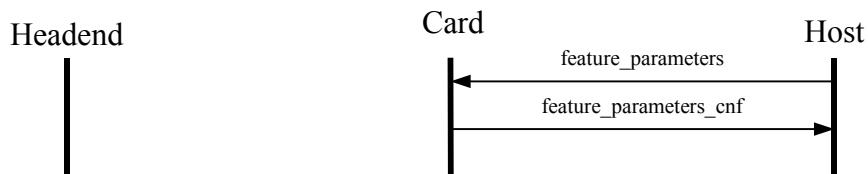
#### 9.15.2.2 Host to CableCARD Device Transfer

After the feature exchange has occurred, the Card MAY request the Host to send its feature parameters (`feature_parameters_req`). After any request, the Host SHALL send to the Card, the parameters for all the generic features in the Host's generic feature list (`feature_parameters`). The Card SHALL reply with the confirmation (`feature_parameters_cnf`). The Card MAY utilize these generic feature parameters, transfer them to the headend, or ignore them.



**Figure 9.15-4 - Host to CableCARD Device Feature Parameters**

When any of the parameters of the generic features that are in the Card generic feature list are changed in the Host, for whatever reason, the Host SHALL transmit these new parameters to the Card (`feature_parameters`). The Card SHALL reply with the confirmation (`feature_parameters_cnf`).



**Figure 9.15-5 - Host Parameter Update**

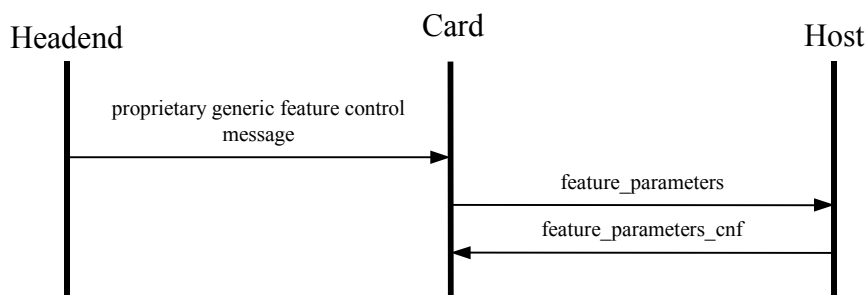
The Card MAY request, at any time the session is open and the generic feature list exchange has occurred, the current parameters in the Host. The Card SHALL do this by sending a *feature\_parameters\_request()* APDU.

### 9.15.2.3 Headend to Host

It is not intended that the headend would transmit all the generic feature's parameters cyclically. Most of the parameters would only be transmitted once at the request of the user or for staging of the device. The generic feature's parameters, which MAY need to be sent cyclically, are the RF output channel, time zone, daylight savings, and rating region. The headend MAY send all or just some of the parameters.

The method in which the Card receives the generic feature's parameters is proprietary to the Card manufacturer.

After the session has been established, when the Card receives a message from the headend containing generic feature parameters, the Card SHALL transfer this information to the Host (*feature\_parameters*). The Host SHALL replace its parameters with the values in the APDU. If the Card utilizes the parameters, it SHALL replace its internal parameters with the values in the message from the headend. The Host SHALL respond with the confirmation (*feature\_parameters\_cnf*). The Host MAY receive parameters for generic features, which it does not support. The Host SHALL ignore any generic feature parameters that it does not implement.



**Figure 9.15-6 - Headend to Host Feature Parameters**

### 9.15.3 Generic Feature Control Resource Identifier

The following resource identifier SHALL be utilized for Generic Feature Control.

**Table 9.15–1 - Generic Feature Control Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic Feature Control	S-Mode/M-Mode	42	1	1	0x002A0041



#### 9.15.4 Feature ID

Each generic feature SHALL have a unique ID assigned to it. This ID is the same for all APDUs. The following is a list of the features and their assigned feature ID.

**Table 9.15–2 - Feature Ids**

Feature ID	Feature
0x00	Reserved
0x01	RF Output Channel
0x02	Parental Control PIN
0x03	Parental Control Settings
0x04	IPPV PIN
0x05	Time Zone
0x06	Daylight Savings Control
0x07	AC Outlet
0x08	Language
0x09	Rating Region
0x0A	Reset PINS
0x0B	Cable URL
0x0C	EAS location code
0x0D-0x6F	Reserved for future use
0x70-0xFF	Reserved for proprietary use

#### 9.15.5 Generic Feature Control APDUs

The Generic Feature Control resource consists of the following 7 APDUs.

**Table 9.15–3 - Generic Feature Control APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ Card
feature_list_req	0x9F9802	Generic Feature Control	↔
feature_list()	0x9F9803	Generic Feature Control	↔
feature_list_cnf()	0x9F9804	Generic Feature Control	↔
feature_list_changed()	0x9F9805	Generic Feature Control	↔
feature_parameters_req()	0x9F9806	Generic Feature Control	↔
feature_parameters()	0x9F9807	Generic Feature Control	↔
feature_parameters_cnf()	0x9F9808	Generic Feature Control	↔

##### 9.15.5.1 feature\_list\_req APDU

The Host SHALL send this APDU to the Card and the Card SHALL send this APDU to the Host to query the generic features that are supported.

**Table 9.15–4 - feature\_list\_req APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>feature_list_req() {     feature_list_req_tag     length_field() }</pre>	24	uimsbf

**feature\_list\_req\_tag**     0x9F9802

### 9.15.5.2 feature\_list APDU

After receiving the *feature\_list\_req()* APDU, the Host or Card SHALL transmit the *feature\_list()* APDU to the Card or Host, which lists the generic features that are supported by the Card or Host.

**Table 9.15–5 - feature\_list APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>feature_list() {     feature_list_tag     length_field()     number_of_features     for (i=0; i&lt;number_of_features; i++) {         feature_id     } }</pre>	24	uimsbf
	8	uimsbf
	8	uimsbf

**feature\_list\_tag**             0x9F9803

**number\_of\_features**     Number of features to report

**feature\_id**                 Assigned feature ID number (See Table 9.15–2).

### 9.15.5.3 feature\_list\_cnf APDU

After receiving the *feature\_list()* APDU, the Host or Card SHALL respond with the *feature\_list\_cnf()* APDU.

**Table 9.15–6 - feature\_list\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
<pre>feature_list_cnf() {     feature_list_cnf_tag     length_field() }</pre>	24	uimsbf

**feature\_list\_cnf\_tag**       0x9F9804

### 9.15.5.4 feature\_list\_changed APDU

The Host or the Card SHALL send the *feature\_list\_changed()* APDU to inform the Card or Host that its feature list has changed.

**Table 9.15–7 - feature\_list\_changed APDU Syntax**

Syntax	No. of Bits	Mnemonic
feature_list_changed() { feature_list_changed_tag length_field() }	24	uimsbf

**feature\_list\_changed\_tag**      0x9F9805

#### **9.15.5.5 feature\_parameters\_req APDU**

After the feature list exchange has occurred, the Card MAY, at any time, send the *feature\_parameters\_req()* APDU to the Host. The Host SHALL NOT send this APDU to the Card.

**Table 9.15–8 - feature\_parameters\_req APDU Syntax**

Syntax	No. of Bits	Mnemonic
feature_paramters_req() { feature_paramters_req_tag length_field() }	24	uimsbf

**feature\_parameters\_req\_tag**      0x9F9806

#### **9.15.5.6 feature\_parameters APDU**

The Host SHALL send the feature\_parameters APDU with its feature list to the Card after receiving a *feature\_parameters\_req()* APDU, or when any of the parameters in the Host's generic feature list are modified, except if the change is the result of receiving a *feature\_parameters()* APDU from the Card. The Card MAY ignore any feature parameters, which it does not support.

The Card MAY send the *feature\_parameters()* APDU at any time in response to a message it receives from the headend.

**Table 9.15–9 - feature\_parameters APDU Syntax**

Syntax	No. of Bits	Mnemonic
feature_parameters() { feature_parameters_tag length_field() number_of_features for (i=0; i<number_of_features; i++) { feature_id if (feature_id == 0x01) { rf_output_channel() } if (feature_id == 0x02) { p_c_pin() } if (feature_id == 0x03) { p_c_settings() } if (feature_id == 0x04) { ippv_pin() } if (feature_id == 0x05) { time_zone() } if (feature_id == 0x06) { daylight_savings() } if (feature_id == 0x07) { ac_outlet() } if (feature_id == 0x08) { language() } if (feature_id == 0x09) { rating_region() } if (feature_id == 0x0A) { reset_pin() } if (feature_id == 0x0B) { cable_urls() } if (feature_id == 0x0C) { EA_location_code() } } }	24  8  8	uimsbf  uimsbf  uimsbf

<b>feature_parameters_tag</b>	0x9F9807
<b>number_of_features</b>	Number of features to report
<b>feature_id</b>	Assigned feature ID number (see Table 9.15–2)
<b>rf_output_channel</b>	RF output channel
<b>p_c_pin</b>	Parental Control PIN parameter
<b>p_c_settings</b>	Parental Control Settings parameter
<b>ippv_pin</b>	IPPV PIN parameter

<b>time_zone</b>	Time Zone parameter <i>This feature is only utilized if the cable system crosses time zones.</i>
<b>daylight_savings</b>	Daylight Savings parameter <i>This feature is only utilized if the cable system encompasses both areas, which recognize daylight savings and those which do not.</i>
<b>ac_outlet</b>	AC Outlet parameter
<b>language</b>	Language parameter
<b>rating_region</b>	Rating Region parameter
<b>reset_pin</b>	Reset PINs
<b>cable_urls</b>	URL list
<b>ea_location_code</b>	EAS location code

#### 9.15.5.7 Feature Parameters Confirmation

Each generic feature will have a parameter definition uniquely assigned. These parameters will be consistent for all APDUs. The following sections define these parameters if the specified features are implemented.

When the Card or Host receives the *feature\_parameter()* APDU, it SHALL respond with the feature parameters confirmation APDU.

**Table 9.15–10 - Feature Parameters Confirm Object Syntax**

Syntax	# of bits	Mnemonic
Feature_parameters_cnf() {		
feature_parameters_cnf_tag	24	uimsbf
length_field()		
number_of_features	8	uimsbf
for(i=0; i<number_of_features; i++){		
feature_id	8	uimsbf
status	8	uimsbf
}		
}		

<b>feature_parameters_tag</b>	Value = 0x9F9808
<b>number_of_features</b>	Number of features to report
<b>feature_ID</b>	Assigned feature ID number as defined in Table 9.15–2
<b>status</b>	Status of feature parameter 0x00 Accepted 0x01 Denied – feature not supported 0x02 Denied – invalid parameter 0x03 Denied – other reason 0x04-0xFF Reserved

**9.15.5.7.1 rf\_output\_channel****Table 9.15–11 - rf\_output\_channel**

Syntax	No. of Bits	Mnemonic
rf_output_channel() { output_channel output_channel_ui }	8 8	uimsbf uimsbf

**output\_channel**

RF output channel. The Host SHALL ignore any value that it cannot accommodate and will use its previous value.

**Output\_channel\_ui**

Enable RF output channel user interface. If disabled, the Host SHALL disable the user from changing the RF output channel.

- 00 Reserved
- 01 Enable RF output channel user interface
- 02 Disable RF output channel user interface
- 03-0xFF Reserved

**9.15.5.7.2 p\_c\_pin****Table 9.15–12 - p\_c\_pin**

Syntax	No. of Bits	Mnemonic
p_c_pin () { p_c_pin_length for (i=0; i<p_c_pin_length; i++) { p_c_pin_chr } }	8 8	uimsbf uimsbf

**p\_c\_pin\_length**

Length of the parental control PIN. Maximum length is 255 bytes.

**p\_c\_pin\_chr**

Parental control PIN character. The value is coded as defined in [ISO10646-1]. The first character received is the first character entered by the user.

**9.15.5.7.3 p\_c\_settings****Table 9.15–13 - p\_c\_settings**

Syntax	No. of Bits	Mnemonic
p_c_settings() { p_c_factory_reset p_c_channel_count for (i=0; i<p_c_channel_count; i++) { reserved major_channel_number minor_channel_number } }	8 16 4 10 10	uimsbf uimsbf '1111' uimsbf uimsbf

**p\_c\_factory\_reset**

Perform factory reset on parental control feature.

0x00-0xA6 No factory reset.  
 0xA7 Perform factory reset.  
 0xA8-0xFF Reserved

**p\_c\_channel\_count** Number of virtual channels to place under parental control

**major\_channel\_number** For two-part channel numbers, this is the major number for a virtual channel to place under parental control. For one-part channel numbers, this is the higher 10 bits of the channel number for a virtual channel to place under parental control. Both two-part and one-part channel numbers SHALL be as defined in [SCTE65].

**minor\_channel\_number** For two-part channel numbers, this is the minor number for a virtual channel to place under parental control. For one-part channel numbers, this is the lower 10 bits of the channel number for a virtual channel to place under parental control. Both two-part and one-part channel numbers SHALL be as defined in [SCTE65].

#### 9.15.5.7.4 purchase\_pin

**Table 9.15–14 - purchase\_pin**

Syntax	No. of Bits	Mnemonic
<code>purchase_pin() {</code>	8	uimsbf
<code>purchase_pin_length</code>		
<code>for (i=0; i&lt;purchase_pin_length; i++) {</code>	8	uimsbf
<code>purchase_pin_chr</code>		
<code>}</code>		
<code>}</code>		

**purchase\_pin\_length** Length of the Purchase PIN. Maximum length is 255 bytes.

**purchase\_pin\_chr** Purchase PIN character. The value is coded as defined in [ISO10646-1]. The first character received is the first character entered by the user.

#### 9.15.5.7.5 time\_zone

**Table 9.15–15 - time\_zone**

Syntax	No. of Bits	Mnemonic
<code>time_zone() {</code>	16	tcimsbf
<code>time_zone_offset</code>		
<code>}</code>		

**time\_zone\_offset** Two's complement integer offset, in number of minutes, from UTC. The value represented SHALL be in the range of –12 to +12 hours. This is intended for systems which cross time zones.

**9.15.5.7.6 daylight\_savings****Table 9.15–16 - daylight\_savings**

Syntax	No. of Bits	Mnemonic
daylight_savings() { daylight_savings_control }	8	uimsbf

**daylight\_savings\_control**      Daylight savings time control

0x00    Ignore this field  
0x01    Do not use daylight savings time  
0x02    Use daylight savings  
0x03-0xFF    Reserved

**9.15.5.7.7 ac\_outlet****Table 9.15–17 - ac\_outlet**

Syntax	No. of Bits	Mnemonic
ac_outlet() { ac_outlet_control }	8	uimsbf

**ac\_outlet\_control**      AC outlet control

0x00    Use user setting  
0x01    Switched AC outlet  
0x02    Unswitched AC outlet (always on)  
0x03-0xFF    Reserved

**9.15.5.7.8 language****Table 9.15–18 - language**

Syntax	No. of Bits	Mnemonic
language() { language_control }	24	uimsbf

**language\_control**      [ISO639-1]: 2002 Codes for the representation of names of Languages – Part 1: Alpha-2 code, and [ISO639-2]: 2002 Codes for the representation of names of Languages – Part 1: Alpha-3 code.



**9.15.5.7.9 rating\_region****Table 9.15–19 - rating\_region**

Syntax	No. of Bits	Mnemonic
rating_region() { rating_region_setting }	8	uimsbf

**rating\_region\_setting**

The 8-bit unsigned integer defined in [SCTE65] that defines the rating region in which the Host resides.

0x00 Forbidden  
 0x01 United States (50 states + possessions)  
 0x02 Canada  
 0x03-0xFF Reserved

**9.15.5.7.10 reset\_pin****Table 9.15–20 - reset\_pin**

Syntax	No. of Bits	Mnemonic
reset_pin() { reset_pin_control }	8	uimsbf

**reset\_pin\_control**

Defines the control of resetting PIN(s). The reset value is defined by the manufacturer and is not covered in this document.

0x00 Do not reset any PIN  
 0x01 Reset parental control PIN  
 0x02 Reset purchase PIN  
 0x03 Reset both parental control and purchase PINs  
 0x04-0xFF Reserved

**9.15.5.7.11 cable\_urls****Table 9.15–21 - cable\_urls**

Syntax	No. of Bits	Mnemonic
cable_urls() { number_of_urls for (i=0; i<number_of_urls; i++) { url_type url_length for (j=0; j<url_length; j++) { url_char } } }	8  8 8 8	uimsbf  uimsbf uimsbf uimsbf

**number\_of\_urls**

Number of URLs defined; used in the following for loop:

**url\_type**

Type of URL

0x00 Undefined

	0x01 Web portal URL
	0x02 EPG URL
	0x03 VOD URL
	0x04-0xFF Reserved
<b>url_length</b>	Length of the URL. Used in the following for loop. The maximum length is 255 bytes.
<b>url_char</b>	A URL character. The restricted set of characters and generic syntax defined in [RFC2396], “Uniform Resource Identifier (URI): Generic Syntax”, SHALL be used.

#### 9.15.5.7.12EA\_location\_code

**Table 9.15–22 - EA\_location\_code**

Syntax	No. of Bits	Mnemonic
EA_location_code() {		
state_code	8	uimsbf
county_subdivision	4	uimsbf
reserved	2	'11'
county_code	10	uimsbf
}		

**state\_code** As defined in [J042]

**county\_subdivision** As defined in [J042]

**county\_code** As defined in [J042]

## 9.16 Generic Diagnostic Support

The Generic Diagnostic Support resource enables the Card to request that the Host perform a diagnostic and report the status/result of the request to the Card. The Card MAY then use the diagnostic information to report diagnostics to the headend or the OSD diagnostic application. If the Card attempts to open a diagnostic support session, and the Host replies that the Generic Diagnostic Support is unavailable, the Card SHALL NOT request any diagnostic information from the Host.

The Card MAY request that the Host perform a diagnostic and report the status/result in response to a headend OOB message, or a SNMP message request to perform a diagnostic that is supported exclusively on the Host.

For M-Mode, this resource has been modified from version 1. Version 2 of this Resource allows for receiving transport stream information from a specific transport stream identified by the Local Transport Stream Identifier (LTSID).

If a Host does support Version 1 of this APDU, the information returned SHALL only be for the primary transport stream (LTSID = 0x01).

**Table 9.16–1 - Generic Diagnostic Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic Diagnostic Support	S-Mode	260	1	1	0x01040041
Generic Diagnostic Support	M-Mode	260	2	1	0x01040081

The Generic Diagnostic Support resource is made up of the following 2 APDUs.

**Table 9.16–2 - Generic Diagnostic Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
diagnostic_req()	0x9FDF00	Generic Diagnostic Support	←
diagnostic_cnf()	0x9FDF01	Generic Diagnostic Support	→

The following values SHALL be used as the diagnostic\_id.

**Table 9.16–3 - Diagnostic Ids**

Diagnostic	Value
Host memory allocation	0x00
Application version number	0x01
Firmware version	0x02
MAC address	0x03
FAT status	0x04
FDC status	0x05
Current Channel Report	0x06
1394 Port	0x07
DVI_status	0x08
eCM	0x09
HDMI Port Status	0x0A
RDC status	0x0B
OCHD2 Network Address	0x0C
Reserved	0x0D-0xFF

### 9.16.1 diagnostic\_req APDU

The Card's diagnostic application SHALL use the *diagnostic\_req()* APDU to request the Host to perform a specific set of diagnostic functions and report the result/status of the diagnostics to the Card's diagnostic application.

**Table 9.16–4 - S-Mode - diagnostic\_req APDU Syntax (Version 1)**

Syntax	No. of Bits	Mnemonic
diagnostic_req() { diagnostic_req_tag length_field() number_of_diag for (i=0; i<number_of_diag; i++) { diagnostic_id } }	24  8  8	uimsbf  uimsbf  uimsbf

**diagnostic\_req\_tag**      0x9FDF00

**number\_of\_diag**      This field indicates the total number of self-diagnostics being requested.

**Diagnostic\_id** This field is a unique ID assigned to a particular diagnostic. These values are defined in Table 9.16–3.

**Table 9.16–5 - M-Mode - diagnostic\_req APDU Syntax (Version 2)**

Syntax	No. of Bits	Mnemonic
diagnostic_req() {		
diagnostic_req_tag	24	uimsbf
length_field()		
number_of_diag	8	uimsbf
for (i=0; i<number_of_diag; i++) {		
diagnostic_id	8	uimsbf
ltsid	8	uimsbf
}		
}		

**diagnostic\_req\_tag** 0x9FDF00

**number\_of\_diag** This field indicates the total number of self-diagnostics being requested.

**Diagnostic\_id** This field is a unique ID assigned to a particular diagnostic. These values are defined in Table 9.16–3.

**Ltsid** Local Transport Stream ID. Only required when the M-CARD is present, and operating in M-Mode.

### 9.16.2 diagnostic\_cnf APDU

The Host SHALL transmit the *diagnostic\_cnf()* APDU after reception of the *diagnostic\_req()* APDU and the Host has completed any tests required to report the result/status.

**Table 9.16–6 - S-Mode - diagnostic\_cnf APDU Syntax (Version 1)**

Syntax	No. of Bits	Mnemonic
diagnostic_cnf() {		
diagnostic_cnf_tag	24	uimsbf
length_field()		
number_of_diag	8	uimsbf
for (i=0; i<number_of_diag; i++) {		
diagnostic_id	8	uimsbf
status_field	8	uimsbf
if (status_field == 0x00) {		
if (diagnostic_id == 0x00) {		
memory_report()		
}		
if (diagnostic_id == 0x01) {		
software_ver_report()		
}		
if (diagnostic_id == 0x02) {		
firmware_ver_report()		
}		
if (diagnostic_id == 0x03) {		
MAC_address_report()		
}		
if (diagnostic_id == 0x04) {		
FAT_status_report()		
}		
if (diagnostic_id == 0x05) {		
FDC_status_report()		
}		
if (diagnostic_id == 0x06) {		
current_channel_report()		
}		
if (diagnostic_id == 0x07) {		
1394_port_report()		
}		
if (diagnostic_id == 0x08) {		
DVI_status_report()		
}		
if (diagnostic_id == 0x09) {		
eCM_status_report()		
}		
if (diagnostic_id == 0x0A) {		
HDMI_port_status_report()		
}		
if (diagnostic_id == 0x0B) {		
RDC_status_report()		
}		
if (diagnostic_id == 0x0C) {		
net_address_report()		
}		
}		
}		

**diagnostic\_cnf\_tag**      0x9FDF01

<b>number_of_diag</b>	This field indicates the total number of self-diagnostics being requested.
<b>Diagnostic_id</b>	This field is a unique ID assigned to a particular diagnostic. These values are defined in Table 9.16–3.
<b>Status_field</b>	Status of the requested diagnostic. See Table 9.16–8.

**Table 9.16–7 - M-Mode - diagnostic\_cnf APDU Syntax (Version 2)**

Syntax	No. of Bits	Mnemonic
diagnostic_cnf() { diagnostic_cnf_tag length_field() number_of_diag for (i=0; i<number_of_diag; i++) { diagnostic_id ltsid status_field if (status_field == 0x00) { if (diagnostic_id == 0x00) { memory_report() } if (diagnostic_id == 0x01) { software_ver_report() } if (diagnostic_id == 0x02) { firmware_ver_report() } if (diagnostic_id == 0x03) { MAC_address_report() } if (diagnostic_id == 0x04) { FAT_status_report() } if (diagnostic_id == 0x05) { FDC_status_report() } if (diagnostic_id == 0x06) { current_channel_report() } if (diagnostic_id == 0x07) { 1394_port_report() } if (diagnostic_id == 0x08) { DVI_status() } if (diagnostic_id == 0x09) { eCM_status_report() } if (diagnostic_id == 0x0A) { HDMI_port_status_report() } if (diagnostic_id == 0x0B) { RDC_status_report() } if (diagnostic_id == 0x0C) { net_address_report() } } } }	24  8  8 8 8	uimsbf  uimsbf uimsbf uimsbf

**diagnostic\_cnf\_tag**      0x9FDF01

**number\_of\_diag**      This field indicates the total number of self-diagnostics being requested.

**Diagnostic\_id**      This field is a unique ID assigned to a particular diagnostic. These values are defined in Table 9.16–3.

**Ltsid** Local Transport Stream ID. Only required when the Card is present, and operating in M-Mode.

**Status\_field** Status of the requested diagnostic. See Table 9.16–8.

**Table 9.16–8 - Table Status Field Values**

Bit Value (Hex)	Status_field
0x00	Diagnostic Granted
0x01	Diagnostic Denied – Feature not Implemented
0x02	Diagnostic Denied – Device Busy
0x03	Diagnostic Denied – Other reasons
0x04-0xFF	Reserved for future use

For Diagnostic\_id values from 0x0D to 0xFF, a Status\_field value of 0x01 SHALL be returned.

### 9.16.3 Diagnostic Report Definition

Each applicable diagnostic SHALL consist of a set of diagnostic reports that SHALL contain a specific set of parameters applicable to the requested diagnostic. The following sections define these reports and their associated parameters.

#### 9.16.3.1 memory\_report

Memory reports SHALL contain the memory parameters associated with the Host.

**Table 9.16–9 - memory\_report**

Syntax	No. of Bits	Mnemonic
memory_report() { number_of_memory if (i=0; i<number_of_memory; i++) { memory_type memory_size } }	8  8 32	uimsbf  uimsbf uimsbf

**number\_of\_memory** The number of memory types being reported in this message.

**memory\_type** Designates the type of memory that is being reported.

0x00 ROM  
0x01 DRAM  
0x02 SRAM  
0x03 Flash  
0x04 NVM  
0x05 Hard drive  
0x06 Video memory  
0x07 Other memory  
0x08-0xFF Reserved

**memory\_size** Designates the physical size of the specified memory type. The units are kilobytes, defined to be 1,024 bytes.



### 9.16.3.2 software\_ver\_report

Software version reports SHALL contain the software version parameters associated with the Host.

**Table 9.16–10 - software\_ver\_report**

Syntax	No. of Bits	Mnemonic
software_ver_report() {		
number_of_applications	8	uimsbf
for (i=0; i<number_of_applications; i++) {		
application_version_number	16	uimsbf
application_status_flag	8	uimsbf
application_name_length	8	uimsbf
for (j=0; j<application_name_length; j++) {		
application_name_byte	8	uimsbf
}		
application_sign_length	8	uimsbf
for (j=0; j<application_sign_length; j++) {		
application_sign_byte	8	uimsbf
}		
}		
}		

**number\_of\_applications** Total number of applications contained with the report.

**application\_version\_number** 16-bit version number of the application.

**application\_status\_flag** Status of the software, either active, inactive or downloading.

0x00 Active  
 0x01 Inactive  
 0x02 Downloading  
 0x03-0xFF Reserved

**application\_name\_length** Designates the number of characters required to define the applications name.

**application\_name\_byte** ASCII character, 8-bits per character, a string that identifies the application.

**application\_sign\_length** Designates the number of characters required to define the application signature.

**application\_sign\_byte** ASCII character, 8-bits per character, a string that identifies the application signature.

### 9.16.3.3 firmware\_ver\_report

Firmware version reports SHALL contain the firmware version parameters associated with the Host.

**Table 9.16–11 - firmware\_ver\_report**

Syntax	No. of Bits	Mnemonic
firmware_ver_report() {		
firmware_version	16	uimsbf
firmware_date{		
firmware_year	16	uimsbf
firmware_month	8	uimsbf
firmware_day	8	uimsbf
}		

**firmware\_version** 16-bit version number of the firmware.

**firmware\_year** 16-bit designation of the firmware's year.

**firmware\_month** 8-bit numerical representation of the firmware's month.

**firmware\_day** 8-bit numerical representation of the firmware's day.

#### 9.16.3.4 MAC\_address\_report

The MAC address report SHALL contain the MAC address parameters associated with the Host.

**Table 9.16–12 - MAC\_address\_report**

Syntax	No. of Bits	Mnemonic
MAC_address_report() { number_of_addresses for (i=0; i<number_of_addresses; i++) { MAC_address_type number_of_bytes for (j=0; j<number_of_bytes; j++) { MAC_address_byte } } }	8  8 8  8	uimsbf  uimsbf uimsbf  uimsbf

**number\_of\_addresses** Total number of MAC addresses contained in the report.

**MAC\_address\_type** Type of device associated with reported MAC address.

0x00 No addressable device available  
0x01 Host  
0x02 1394 port  
0x03 Reserved  
0x04 DOCSIS  
0x05 Reserved  
0x06-0xFF Reserved

**number\_of\_bytes** The total number of bytes required for the MAC address.

**MAC\_address\_byte** One of a number of bytes that constitute the Media Access Control (MAC) address of the Host device. Each byte represents 2 hexadecimal values (xx) in the range of 0x00 to 0xFF.

#### 9.16.3.5 FAT\_status\_report

In response to a FAT status report request, the Host SHALL reply with a FAT\_status\_report, unless an error has occurred.

**Table 9.16–13 - FAT\_status\_report**

Syntax	No. of Bits	Mnemonic
FAT_status_report() { reserved PCR_lock modulation_mode carrier_lock_status SNR signal_level }	4 1 2 1 16 16	'1111' bslbf bslbf bslbf tcimsbf tcimsbf

**PCR\_lock** Indicates if the FAT channel receiver is locked to the currently tuned channel  
0b Not locked

	1b	Locked
<b>modulation_mode</b>	Indicates if the current forward transport is analog, QAM-64, or QAM-256	
	00b	Analog
	01b	QAM64
	10b	QAM256
	11b	Reserved
<b>carrier_lock_status</b>	Indicates if the current carrier is locked or not locked.	
	0b	Not locked
	1b	Locked
<b>SNR</b>	Numerical representation of the signal to noise ratio in tenths of a dB.	
<b>signal_level</b>	Numerical representation of the signal level in tenths of a dBmV.	

### 9.16.3.6 FDC\_status\_report

In response to a FDC status report request, the Host SHALL reply with a FDC status report, unless an error has occurred.

**Table 9.16–14 - FDC\_status\_report**

Syntax	No. of Bits	Mnemonic
FDC_report() {		
FDC_center_freq	16	uimsbf
reserved	6	'111111'
carrier_lock_status	1	bslbf
packet_sync_status	1	bslbf
}		

**FDC\_center\_freq** Indicates the frequency of the FDC center frequency, in MHz. (Frequency = value \* 0.05 + 50 MHz).

**Table 9.16–15 - FDC Center Frequency Value**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Frequency (MS)										Frequency (LS)					

<b>carrier_lock_status</b>	Indicates if the current carrier is locked or not locked.	
	0b	Not locked
	1b	Locked
<b>packet_sync_status</b>	Indicates if the current FDC packets are in sync.	
	0b	Not in sync
	1b	In sync

### 9.16.3.7 current\_channel\_report

In response to a Current Channel report request, the Host SHALL reply with a current\_channel report, unless an error has occurred.

**Table 9.16–16 - current\_channel\_report**

Syntax	No. of Bits	Mnemonic
current_channel() { Reserved channel_type authorization_flag purchasable_flag purchased_flag preview_flag parental_control_flag current_channel }	 2 1 1 1 1 1 1 16	 '11' bslbf bslbf bslbf bslbf bslbf bslbf uimsbf

<b>channel_type</b>	Indicates if the channel is analog or digital. 0b Analog 1b Digital
<b>authorization_flag</b>	Indicates if the Host is authorized for the currently tuned channel. 0b Not authorized 1b Authorized
<b>purchasable_flag</b>	Indicates if the currently tuned channel MAY be purchased. 0b Not purchasable 1b Purchasable
<b>purchased_flag</b>	Indicates if the currently tuned channel has been purchased. 0b Not purchased 1b Purchased
<b>preview_flag</b>	Indicates if the currently tuned channel is in preview mode. 0b Not in preview mode 1b In preview mode
<b>parental_control_flag</b>	Indicates if the currently tuned channel is under parental control. 0b Channel is not blocked 1b Channel is blocked
<b>current_channel</b>	Indicates the numerical representation of the currently tuned channel.

**9.16.3.8 1394\_port\_report**

In response to a 1394 Port report request, the Host SHALL reply with a 1394\_port\_report, unless an error has occurred.

**Table 9.16–17 - 1394\_port\_report**

Syntax	No. of Bits	Mnemonic
1394_port_report() { Reserved loop_status root_status cycle_master_status a/d_source_selection_status port_1_connection_status port_2_connection_status total_number_of_nodes }	 3 1 1 1 1 1 1 16	 '111' bslbf bslbf bslbf bslbf bslbf bslbf uimsbf

<b>loop_status</b>	Indicates if a loop exists on the 1394 bus. 0b No loop exists 1b Loop exists
<b>root_status</b>	Indicates if the Host device is the root node on the 1394 bus. 0b Not root 1b Is root
<b>cycle_master_status</b>	Indicates if the Host device is the cycle master node on the 1394 bus. 0b Not cycle master 1b Is cycle master
<b>a/d_source_selection_status</b>	Indicates if the Host device supports A/D source selection function. 0b Not Supported 1b Supported
<b>port_1_connection_status</b>	Indicates if port 1 of the 1394 PHY is connected to a 1394 bus. 0b Not connected 1b Connected
<b>port_2_connection_status</b>	Indicates if port 2 of the 1394 PHY is connected to a 1394 bus. 0b Not connected 1b Connected
<b>total_number_of_nodes</b>	Indicates the total number of nodes connected to the 1394 bus. A maximum of 65,535 nodes MAY exist, excluding the Host (a maximum of 64 nodes with a maximum of 1,024).

### 9.16.3.9 DVI Status Report

In response to a DVI Status Report request, the Host SHALL reply with a DVI Status Report, unless an error has occurred.

**Table 9.16–18 - DVI Status Report Syntax**

Syntax	# of bits	Mnemonic
DVI_status_report() { reserved connection_status host_HDCP_status device_HDCP_status video_format { horizontal_lines vertical_lines frame_rate aspect_ratio prog_inter_type reserved } }	3 2 1 2  16 16 8 2 1 5	'111' bslbf bslbf bslbf  uimsbf uimsbf uimsbf bslbf bslbf bslbf

<b>connection_status</b>	Indicates if a connection exists on the DVI port 00b No connection exists 01b Device connected – not repeater 10b Device connected – repeater 11b Reserved
<b>host_HDCP_status</b>	Indicates if HDCP is enabled on the DVI link

	0b	Not enabled
	1b	Enabled
<b>device_HDCP_status</b>	Indicates the connected device's HDCP status (valid only when connection_status is not equal to 00 <sub>2</sub> ).	
	00b	Non HDCP device
	01b	Compliant HDCP device
	10b	Revoked HDCP device
	11b	Reserved
<b>video_format</b>	Indicates the current video format utilized on the DVI port as defined in the following fields:	
<b>horizontal_lines</b>	Indicates the number of horizontal lines associated with the video format on the DVI link.	
<b>vertical_lines</b>	Indicates the number of vertical lines associated with the video format on the DVI link.	
<b>frame_rate</b>	Indicates the frame rate associated with the video format on the DVI link as defined in the following table.	

**Table 9.16–19 - Frame Rate Associated With the Video Format On the DVI Link**

Frame Rate Code	Frame Rate
01	23.976 Hz
02	24 Hz
04	29.97 Hz
05	30 Hz
07	59.94 Hz
08	60 Hz

<b>aspect_ratio</b>	Indicates the aspect ratio associated with the video format on the DVI link as defined in the following table:
---------------------	--

**Table 9.16–20 - Aspect Ratio Associated With the Video Format On the DVI Link**

Bit Value	Video Format
00	4:3
01	16:9
10	Reserved
11	Reserved

<b>prog_inter_type</b>	Indicates if the video is progressive or interlaced on the DVI link,
	0b Interlaced
	1b Progressive

**9.16.3.10 eCM Status Report**

In response to an embedded cable modem status request report, the Host SHALL reply with an eCM status report, unless an error has occurred.

**Table 9.16–21 - eCMStatus Report Syntax**

Syntax	# of bits	Mnemonic
eCM_status_report() { reserved downstream_center_freq downstream_power_level downstream_carrier_lock_status upstream_xmt_center_freq upstream_power_level upstream_symbol_rate upstream_modulation_type }	6 16 16 1 16 16 8 1	"111111" uimsbf tcimsbf bslbf uimsbf tcimsbf uimsbf bslbf

**downstream\_center\_freq** Indicates the frequency of the FDC center frequency, in MHz  
(Frequency = value \* 0.05 + 50 MHz).

**Table 9.16–22 - Downstream Center Frequency Value**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Frequency (MS)								Frequency (LS)							

**downstream\_power\_level** Numerical representation of the signal level in tenths of a dBmV.

**carrier\_lock\_status** Indicates if the current carrier is locked or not locked.

0b Not locked

1b Locked

**upstream\_xmt\_center\_freq** Indicates the frequency of the RDC center frequency, in MHz  
(Frequency = value \* 0.05 + 5 MHz).

**Table 9.16–23 - Upstream Transmit Center Frequency Value**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Frequency (MS)								Frequency (LS)							

**upstream\_power\_level** Numerical representation of the signal level in dBmV.

**upstream\_symbol\_rate** Numerical representation of the symbol rate as defined below.

0x00 0.16 Msps

0x01 0.32 Msps

0x02 0.64 Msps

0x03 1.28 Msps

0x04 2.56 Msps

0x05 - 0xFF Reserved

**upstream\_modulation\_type** Indicates the current modulation type.

0b QPSK

1b 16QAM

### 9.16.3.11 HDMI Port Status Report

In response to a HDMI Status Report request, the Host SHALL reply with a HDMI Status Report, unless an error has occurred.

**Table 9.16–24 - HDMI Status Report Syntax**

Syntax	# of bits	Mnemonic
HDMI_status_report() {		
Reserved	3	'111'
connection_status	2	bslbf
host_HDMI_status	1	bslbf
device_HDCP_status	2	bslbf
video_format		
{		
horizontal_lines	16	uimsbf
vertical_lines	16	uimsbf
frame_rate	8	uimsbf
aspect_ratio	2	bslbf
prog_inter_type	1	bslbf
reserved	5	bslbf
}		
}		

#### connection\_status

Indicates if a connection exists on the HDMI port

- 00b No connection exists
- 01b Device connected, no repeater
- 10b Device connected, with repeater
- 11b Reserved

#### host\_HDCP\_status

Indicates if HDCP is enabled on the HDMI link

- 0b Not enabled
- 1b Enabled.

#### device\_HDCP\_status

Indicates the connected device's HDCP status (valid only when connection\_status is not equal to 00<sub>2</sub>)

- 00b Non HDCP device
- 01b Compliant HDCP device
- 10b Revoked HDCP device
- 11b Reserved

#### video\_format

Indicates the current video format utilized on the HDMI port as defined in the following fields:

#### horizontal\_lines

Indicates the number of horizontal lines associated with the video format on the HDMI link.

#### vertical\_lines

Indicates the number of vertical lines associated with the video format on the HDMI link.

#### frame\_rate

Indicates the frame rate associated with the video format on the HDMI link as defined in the following table.

**Table 9.16–25 - Frame Rate Associated With the Video Format On the HDMI Link**

Frame Rate Code	Frame Rate
01	23.976 Hz
02	24 Hz



Frame Rate Code	Frame Rate
04	29.97 Hz
05	30 Hz
07	59.94 Hz
08	60 Hz

**aspect\_ratio**

Indicates the aspect ratio associated with the video format on the HDMI link as defined in the following table:

**Table 9.16–26 - Aspect Ratio Associated With the Video Format On the HDMI Link**

Bit Value	Video Format
00	4:3
01	16:9
10	Reserved
11	Reserved

**prog\_inter\_type**

Indicates if the video is progressive or interlaced on the HDMI link,

0b Interlaced  
1b Progressive

**9.16.3.12 RDC Status Report**

In response to a RDC status report request, the Host SHALL reply with a RDC status report, unless an error has occurred

**Table 9.16–27 - RDC\_status\_report**

Syntax	No. of Bits	Mnemonic
RDC_report() {		
RDC_center_freq	16	uimsbf
reserved	6	'111111'
RDC_transmitter_power_level	8	tcimsbf
RDC_data_rate	2	bslbf
}		

**RDC\_center\_freq**

Indicates the frequency of the RDC center frequency, in MHz  
(Frequency = value \* 0.05 + 5 MHz).

**Table 9.16–28 - RDC Center Frequency Value**

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Frequency (MS)								Frequency (LS)							

**RDC\_transmitter\_power\_level** Indicates the RDC power level in dBmV.

**RDC\_data\_rate**

Indicates the current RDC data rate.

00b 256kbps  
01b 1544kbps  
10b 3088kbps  
11b Reserved

### 9.16.3.13 OCHD2 Network Address

The `net_address_report` SHALL contain the network address parameters associated with the Host and the Card.

**Table 9.16–29 - `net_address_report`**

Syntax	No. of Bits	Mnemonic
<code>net_address_report() {</code>		
<code>number_of_addresses</code>	8	<code>uimsbf</code>
<code>for (i=0; i&lt;number_of_addresses; i++) {</code>		
<code>net_address_type</code>	8	<code>uimsbf</code>
<code>number_of_bytes</code>	8	<code>uimsbf</code>
<code>for (j=0; j&lt;number_of_bytes; j++) {</code>		
<code>net_address_byte</code>	8	<code>uimsbf</code>
<code>for (j=0; j&lt;number_of_bytes; j++) {</code>		
<code>sub_net_address_byte</code>	8	<code>uimsbf</code>
<code>}</code>		
<code>}</code>		
<code>}</code>		
<code>}</code>		

**number\_of\_addresses** Total number of network addresses contained in the report.

**net\_address\_type** Type of device associated with reported network address.

0x00 No addressable device available  
0x01 Host  
0x02 1394 port  
0x03 Reserved  
0x04 DOCSIS  
0x05 Reserved  
0x06 CableCARD  
0x07-0xFF Reserved

**number\_of\_bytes** The total number of bytes required for the network address.

**net\_address\_byte** One of a number of bytes that constitute the Network addresses assigned to the Host device. Each byte represents 2 hexadecimal values (xx) in the range of 0x00 to 0xFF.

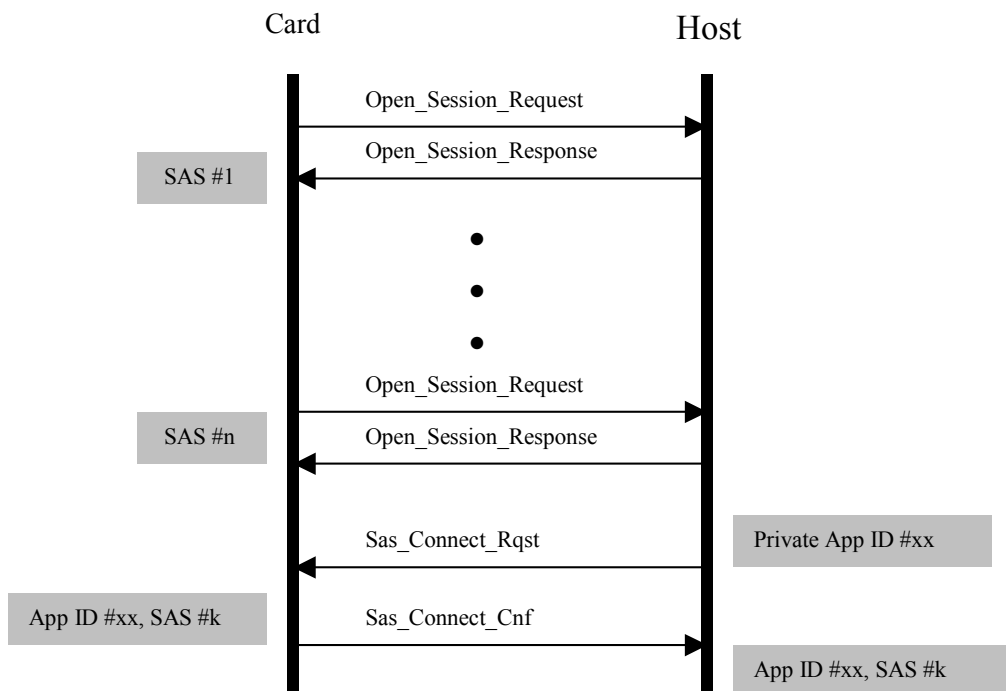
**sub\_net\_address\_byte** One of a number of bytes that constitute the Network addresses assigned to the Host device. Each byte represents 2 hexadecimal values (xx) in the range of 0x00 to 0xFF.

## 9.17 Specific Application Support

The *Specific Application Support* resource is intended for use when a vendor-specific application, which resides in either the Card or the Host, needs to communicate a private set of objects across the interface. Support for this resource is required in the Host and Card. The Card SHALL establish at least one session for communication with the Specific Application Support Resource. Private Host applications and a corresponding specific application in the Card MAY use one of two possible modes of communication.

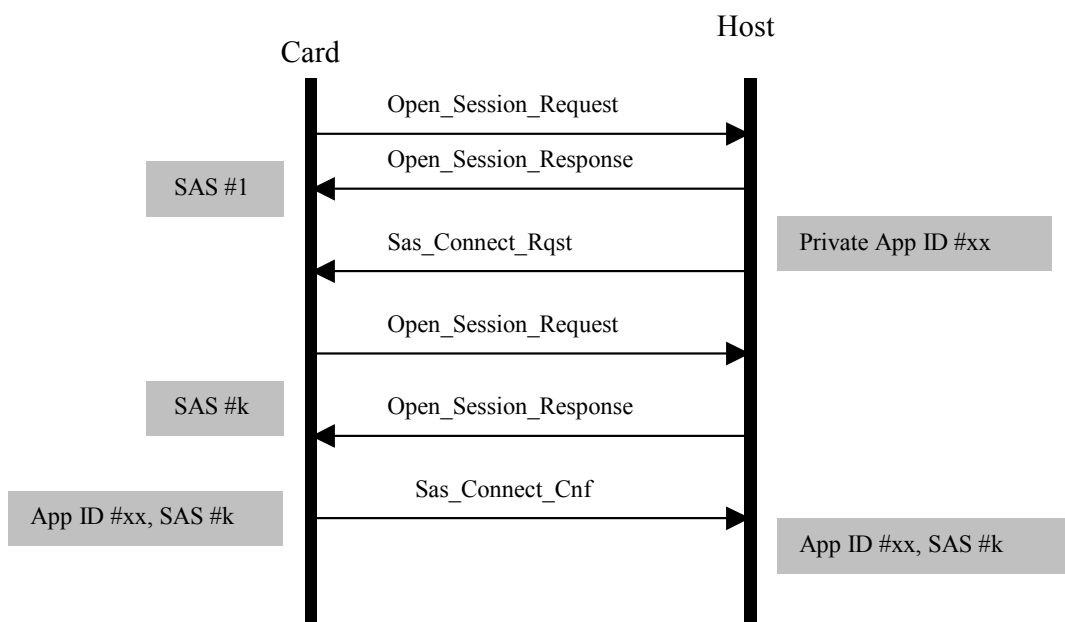
- Synchronous mode: where either the *SAS\_data\_rqst()*, *SAS\_data\_av()*, *SAS\_data\_cnf()*, *SAS\_server\_query()* and *SAS\_server\_reply()* group of APDUs are used to communicate a private set of objects across the interface where flow control is managed at the APDU level.
- Asynchronous mode: where only the *SAS\_async\_msg()* APDU is used to communicate a private set of objects across the interface where flow control is managed at the vendor-specific application level between Host and Card applications.

The CableCARD device MAY open one or more Specific Application Support (SAS) sessions for private communications between vendor-specific Card applications and private Host applications. The Card, as the initiator of the sessions, is responsible for associating each session (by session number) with the appropriate vendor-specific Card application. When a private Host application is ready to establish a connection with the Card, a *SAS\_connect\_rqst()* APDU is sent to the Card over any opened SAS session. The Card uses the private Host application ID to identify the specific SAS session that SHOULD be used for communication between the identified private Host application and the appropriate vendor-specific Card application. This private Host application ID is returned to the Host via the *SAS\_connect\_cnf()* APDU. This operation establishes the communication path between a specific pair of applications (vendor-specific Card application, private Host application).



**Figure 9.17-1 - Specific Application Support Connection Sequence**

In some instances, the Card MAY receive a *SAS\_connect\_rqst()* APDU before a session has been opened for the associated vendor-specific application. In this case, the Card SHALL establish the necessary SAS session and then respond with the *SAS\_connect\_cnf()* APDU.



**Figure 9.17-2 - Specific Application Support Alternate Connection Sequence**

**Table 9.17–1 - Specific Application Support Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Specific Application Support	S-Mode/M-Mode	144	1	2	0x00900042

The Specific Application Support resource includes seven APDUs as described in the following table:

**Table 9.17–2 - Specific Application Support APDUs**

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
SAS_connect_rqst()	0x9F9A00	Specific Application Support	→
SAS_connect()	0x9F9A01	Specific Application Support	←
SAS_data_rqst()	0x9F9A02	Specific Application Support	↔
SAS_data_av()	0x9F9A03	Specific Application Support	↔
SAS_data_cnf()	0x9F9A04	Specific Application Support	↔
SAS_data_query()	0x9F9A05	Specific Application Support	↔
SAS_data_reply()	0x9F9A06	Specific Application Support	↔
SAS_async_msg()	0x9F9A07	Specific Application Support	↔

### 9.17.1 SAS\_connect\_rqst APDU

If required by a private Host application, the Host SHALL send a **SAS\_connect\_rqst()** APDU to the Card to establish a connection between that application and the corresponding Card vendor-specific application.

**Table 9.17–3 - SAS\_connect\_rqst APDU Syntax**

Syntax	No. of Bits	Mnemonic
SAS_connect_rqst () { SAS_connect_rqst_tag length_field() private_host_application_ID }	24  64	uimsbf  uimsbf

**SAS\_connect\_rqst\_tag**           0x9F9A00

**private\_host\_application\_ID**   This is a unique identifier of the private Host application.

**NOTE (Informative):** There is no need to register private\_host\_application\_id used by different manufacturers. Applications that make use of this resource are downloaded into the Host by the cable operator, and thus the application has knowledge of valid ID values that are expected from operator-supplied Cards.

### 9.17.2 SAS\_connect\_cnf APDU

After receiving the *SAS\_connect\_rqst()* APDU, the Card SHALL reply with a *SAS\_connect\_cnf()* APDU to inform the Host of which SAS session is to be used for this connection.

**Table 9.17–4 - SAS\_connect\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
SAS_connect_cnf() { SAS_connect_cnf_tag length_field() private_host_application_ID SAS_session_status }	24  64 8	uimsbf  uimsbf uimsbf

**SAS\_connect\_cnf\_tag**           0x9F9A01

**private\_host\_application\_ID**   This is a unique identifier of the private Host application.

**NOTE (Informative):** There is no need to register private\_host\_application\_id used by different manufacturers. Applications that make use of this resource are downloaded into the Host by the cable operator, and thus the application has knowledge of valid ID values that are expected from operator-supplied Cards.

**SAS\_session\_status**           The status of the requested connection.

0x00   Connection established  
0x01   Connection denied – no associated vendor-specific Card application found  
0x02   Connection denied – no more connections available  
0x03-0xFF   Reserved

### 9.17.3 SAS\_data\_rqst APDU

Once a communication path has been established between the application pair (vendor-specific Card application and private Host application), via a SAS session, each of the applications can utilize the *SAS\_data\_rqst()* APDU to inform the other application that it is ready to process incoming data as well as request data from the other application. This APDU is bidirectional in that it can originate from either side of the CHI. When an application receives a *SAS\_sata\_req()* APDU, it SHALL understand that the sending application is ready to process incoming data for the remainder of the SAS connection's lifetime. Although an application needs to send only one

*SAS\_data\_req()* for the life of a connection, it MAY send more than one *SAS\_data\_req()* APDU over the lifetime of a SAS connection.

**Table 9.17-5 - SAS\_data\_rqst APDU Syntax**

Syntax	No. of Bits	Mnemonic
<i>SAS_data_rqst()</i> { <i>SAS_data_rqst_tag</i> <i>length_field()</i> }	24	uimsbf

**SAS\_data\_rqst\_tag**                      0x9F9A02

#### 9.17.4 SAS\_data\_av APDU

Once a communication path has been established between the application pair (vendor-specific Card application and private Host application) via a SAS session, each of the applications can utilize the *SAS\_data\_av()* APDU to indicate when the application has data to send across the CHI. An application SHALL NOT send *SAS\_data\_av()* without first receiving a *SAS\_data\_rqst()* APDU.

Note: The data itself is transmitted in the *SAS\_query()* and *SAS\_reply()* APDUs.

**Table 9.17-6 - SAS\_data\_av APDU Syntax**

Syntax	No. of Bits	Mnemonic
<i>SAS_data_av()</i> { <i>SAS_data_av_tag</i> <i>length_field()</i> <i>SAS_data_status</i> <i>transaction_nb</i> }	24  8 8	uimsbf  uimsbf uimsbf

**SAS\_data\_av\_tag**                      0x9F9A03

**SAS\_data\_status**                      Status of the available data.  
   0x00    Data available  
   0x01    Data not available  
   0x02-0xFF    Reserved

**transaction\_nb**                      The transaction number is issued from an 8-bit cyclic counter (1-255) and is used to identify each data transaction and to gain access to the available data. When data is not available, the *transaction\_nb* SHALL be set to 0x00.

#### 9.17.5 SAS\_data\_cnf APDU

Once a communication path has been established between the application pair (vendor-specific Card application and private Host application), via a SAS session, after an application receives a *SAS\_data\_av()* APDU, the application SHALL transmit the *SAS\_data\_cnf()* APDU to acknowledge that it is preparing to receive the available data.

**Table 9.17-7 - SAS\_data\_cnf APDU Syntax**

Syntax	No. of Bits	Mnemonic
<i>SAS_data_av_cnf()</i> { <i>SAS_data_av_cnf_tag</i> <i>length_field()</i> <i>transaction_nb</i> }	24  8	uimsbf  uimsbf

**SAS\_data\_av\_cnf\_tag** 0x9F9A04

**transaction\_nb** The transaction number is issued from an 8-bit cyclic counter (1-255) and is used to identify each data transaction and to gain access to the available data. When data is not available, the transaction\_nb SHALL be set to 0x00.

### 9.17.6 SAS\_server\_query APDU

When data availability has been confirmed, a *SAS\_server\_query()* APDU SHALL be sent to initiate the transfer of application specific data.

**Table 9.17-8 - SAS\_server\_query APDU Syntax**

Syntax	No. of Bits	Mnemonic
SAS_server_query () { SAS_server_query_tag	24	uimsbf
length_field() transaction_nb }	8	uimsbf

**SAS\_server\_query\_tag** 0x9F9A05

**transaction\_nb** The transaction\_nb assigned in the *SAS\_data\_av()* APDU.

### 9.17.7 SAS\_server\_reply APDU

After receiving the *SAS\_server\_query()* APDU, the application SHALL respond with the *SAS\_server\_reply()* APDU with the data to transfer.

**Table 9.17-9 - SAS\_server\_reply APDU Syntax**

Syntax	No. of Bits	Mnemonic
SAS_server_reply() { SAS_server_reply_tag	24	uimsbf
length_field() transaction_nb	8	uimsbf
message_length	16	uimsbf
for (i=0; i<message_length; i++) { message_byte	8	uimsbf
} }		

**SAS\_server\_reply\_tag** 0x9F9A06

**transaction\_nb** The transaction\_nb assigned in the *SAS\_data\_av()* APDU.

**message\_length** The length of the message in the following for loop.

**message\_byte** The data to transfer.

### 9.17.8 SAS Async APDU

The *sas\_async\_msg()* APDU may be used, instead of *sas\_data\_rqst()*, *sas\_data\_av()*, *sas\_data\_cnf()*, *sas\_server\_query()* and *sas\_server\_reply()* group of APDUs in order to reduce the overhead and the time needed to send a message to/from vendor-specific applications. Once a communication path has been established between the application pair (vendor-specific Card application, Private Host application) via an SAS session, each of the applications can utilize the *sas\_async\_msg()* APDU to communicate with the other. The *sas\_async\_msg()* APDU

is bi-directional and can originate from either side of the CHI. It is the responsibility of the applications to take care of overflow prevention and ensure reliable delivery of messages.

**Table 9.17–10 - SAS\_Async Message APDU Syntax**

Syntax	No. of Bits	Mnemonic
<code>SAS_async_msg() {</code>		
<code>SAS_async_msg_tag</code>	24	uimsbf
<code>length_field()</code>		
<code>message_nb</code>	8	uimsbf
<code>message_length</code>	16	uimsbf
<code>for (i =0; i&lt; message_length; i++)</code>		
<code>}</code>		
<code>message_byte</code>	8	uimsbf

**SAS\_async\_msg\_tag** 0x9F9A07

**message\_nb:** The message number is issued from an 8-bit cyclic counter (0 – 255) and is used to identify each message.

**message\_length:** The number of bytes in a message.

**message\_bytes:** The message payload in a format agreed between a private Host application and corresponding specific Card application.

## 9.18 Card Firmware Upgrade

The Card SHALL support firmware upgrades as defined by this document.

### 9.18.1 Introduction

The Card will require that its firmware be upgraded occasionally. The mechanism of upgrading this firmware is unique to each Card manufacturer's system. This operation MAY be facilitated by adding the interface outlined in this section. New versions of the Homing and Host Control resources are utilized which encapsulates the previous operations of the resources but adds new operations for facilitating the firmware upgrade.

#### 9.18.1.1 Summary

##### 9.18.1.1.1 Firmware Upgrade

A Card MAY be designed to be capable of having its firmware reprogrammed. Generally, this is implemented with flash memory or battery backed up RAM. Occasionally, this firmware will be upgraded. There are generally two paths in which the firmware can be upgraded: 1) over the cable network using the QAM inband channel, and 2) over the cable network using the QPSK OOB or DSG channel. Upgrade can be accomplished either by the methods defined in this document or by other methods. Since different system implementations affect the method of Card upgrade, two types of upgrade states are offered, a “delayed” and an “immediate”.

##### 9.18.1.1.1.1 Delayed Upgrade

When the Card detects that a firmware upgrade is required and immediate upgrade has not been requested by the headend, then if the Homing resource is not already open, and the Card requires utilizing the Homing resource, it will open a session to the Homing resource if it is not already open. The Card will then wait until the *open\_homing()* APDU is received prior to beginning the upgrade. The Card will inform the Host through the *firmware\_upgrade()* APDU that it will be doing a firmware upgrade. After receiving the *firmware\_upgrade\_reply()* APDU, the Card can use the Host Control resource to tune either the QAM or QPSK tuner in the Host to the appropriate frequency and modulation type. The Host will not modify the selected tuner until the Card has indicated that the firmware upgrade has finished by sending the *firmware\_upgrade\_complete()* APDU or a timeout condition occurs. The *firmware\_upgrade\_complete()* APDU can also indicate to the Host



whether a PCMCIA reset, Card reset, or no reset is required by the Card. After receiving the *firmware\_upgrade\_complete()* APDU, the Host will be free to change the QAM tuner.

The Host will send the *open\_homing()* APDU when it is in standby mode (power applied but in the "Off" state) as defined in [NRSSB].

#### 9.18.1.1.1.2 Immediate Upgrade

There are conditions in which the Card will need to perform an immediate upgrade. When this is required, the Card will have the option to use the interface upgrade mechanisms defined in this document. If using these mechanisms, the Card will open the Homing resource, if it is not already open, and send a *firmware\_upgrade()* APDU. The Host will reply with a *firmware\_upgrade\_reply()* when it is ready. The Card will use the Host Control APDUs to tune either the QAM or QPSK tuner in the Host to the appropriate frequency and modulation type. The Host will not interrupt this process until it has either received a *firmware\_upgrade\_complete()* APDU or a timeout condition occurs. An optional text message is included in the APDU to display to the user if the Host is not in standby.

Additionally, it is possible that an outside occurrence, such as a power failure, may cause the firmware to become corrupted. If this occurs, then the Card is incapable of performing most of its functions. It is still able to perform some functions if ROM code is included in the design. Generally, this ROM code is fairly small since it is not upgradeable and is utilized only for verification of the firmware and loading the firmware in case of corruption. This ROM code, called bootloader code in this document, SHALL be carefully designed and verified since it cannot be modified.

The bootloader is called upon reset of the Card CPU. It first performs basic initialization operations, then tests the main program memory to insure that it is valid, and if it is valid, starts executing out of the main firmware memory. The problem occurs that if the main program memory is not valid, then a mechanism is needed to allow for recovery of the main firmware.

For this rare condition, the bootloader will contain firmware, which will allow the Card to utilize the APDUs defined in this document for an immediate upgrade.

#### 9.18.1.1.2 Inband Upgrade Considerations

If the Card utilizes the QAM inband channel for upgrades, then for normal upgrades it should utilize the delayed upgrade. The Host should then notify the Card that it can upgrade when the Host is placed in the standby state by the user. If the Host has been in the on state for a long period of time or the Card bootloader has detected corrupted memory, then an immediate upgrade is required in which case the Host will give control of the QAM tuner immediately to the Card, independent of its state.

#### 9.18.1.1.3 OOB Upgrade Considerations

If the Card utilizes the QPSK OOB or DSG channel for upgrades, then its operation will depend on whether applications can still operate while performing an upgrade. If they cannot, a delayed firmware upgrade should be used. The Card will have to open the Homing resource and wait until the *open\_homing()* APDU is received prior to beginning the upgrade. If applications can operate during an upgrade, then an immediate firmware upgrade can be used.

#### 9.18.1.1.4 Other Homing Operations

If desired, the Card can use the Homing resource for receiving other parameters over the inband channel when the Host is in standby state. If this is utilized, then the upgrade option should not be used so as to allow the Host to return to the on state at the users request.

## 9.18.2 Implementation

### 9.18.2.1 Introduction

In order to meet these operations, there is a need for a mechanism whereby the Card can inform the Host that a firmware upgrade is required, an optional text message to the user, and the type of upgrade path.

Note that it is the responsibility of the Host to inform the user when an immediate upgrade occurs and to determine when the recovery can occur for delayed upgrades.

### 9.18.2.2 Reset Implementation (Normative)

After the Card has finished its firmware upgrade, it will either send the *firmware\_upgrade\_complete()* APDU with the appropriate reset type or simply timeout based on the timeout type.

## 9.18.3 Host Operation (Normative)

While the Card is performing its upgrade operation, its ability to support the normal Card interface may range from severely limited to entirely unimpaired. To accommodate any case, some modifications to normal operation are required. The following is a list of those modifications as well as requirements to the Host.

1. If enabled by the *firmware\_upgrade()* APDU, the Card SHALL still respond to the transport layer polls with a 5-second timeout. If the Card fails to respond to the poll within 5 seconds, the Host SHALL perform a PCMCIA reset on the Card.
2. The Card may not be able to support session or application layer operations. The Host SHALL NOT initiate any new sessions or any application layer operations after receiving a *firmware\_upgrade()* APDU until either the *firmware\_upgrade\_complete()* APDU is received or the Card times out. However, the Host SHALL maintain all session connections so that if the Card cancels the firmware upgrade, normal operation can continue.
3. The Card may be fully able to support session or application layer operations while performing a firmware upgrade. The Host SHALL respond to any session or application layer operation initiated by the Card and perform all descendant operations consistent with normal Host-Card interface. This includes timeout and reset operation and the initiation of required session and application layer operations.
4. If the download\_timeout\_period expires, the Host SHALL perform a PCMCIA reset on the Card.

If the Card sends a *firmware\_upgrade\_complete()* APDU with No Reset Required, then the Host SHALL resume normal operation with the Card in all respects, including timeout and reset operation.

### 9.18.3.1.1 Timeout Types

The *firmware\_upgrade()* APDU includes a variable called timeout\_type, which defines the type of timeout the Host is to utilize during a firmware upgrade. This can include the normal 5-second transport timeout and/or a download timeout timer, which starts from the last *firmware\_upgrade()* APDU received or neither. It is highly recommended that the Card not use the “No timeout” option.

### 9.18.3.1.2 Transport Layer Timeout

Since the Card may be incorporating flash memory which takes a longer time to program than the transport layer timeout period (5 seconds), using option 02 or 03 on the timeout\_type variable in the *firmware\_upgrade()* APDU will cause the Host to cease implementing this timeout until either a *firmware\_upgrade\_complete()* APDU is received or the download\_timeout\_period from the last *firmware\_upgrade()* APDU has passed, in which case the Host will perform a PCMCIA reset.

### **9.18.3.2 Upgrade Cancellation**

If the Card cancels its firmware upgrade, then it will send the *firmware\_upgrade\_complete()* APDU with the reset type set to 02, “no reset required”.

### **9.18.3.3 Flowchart (Informative)**

Figure 9.18-1 is a flowchart which shows the Card/Host interface which uses Card upgrade methods defined in this document.

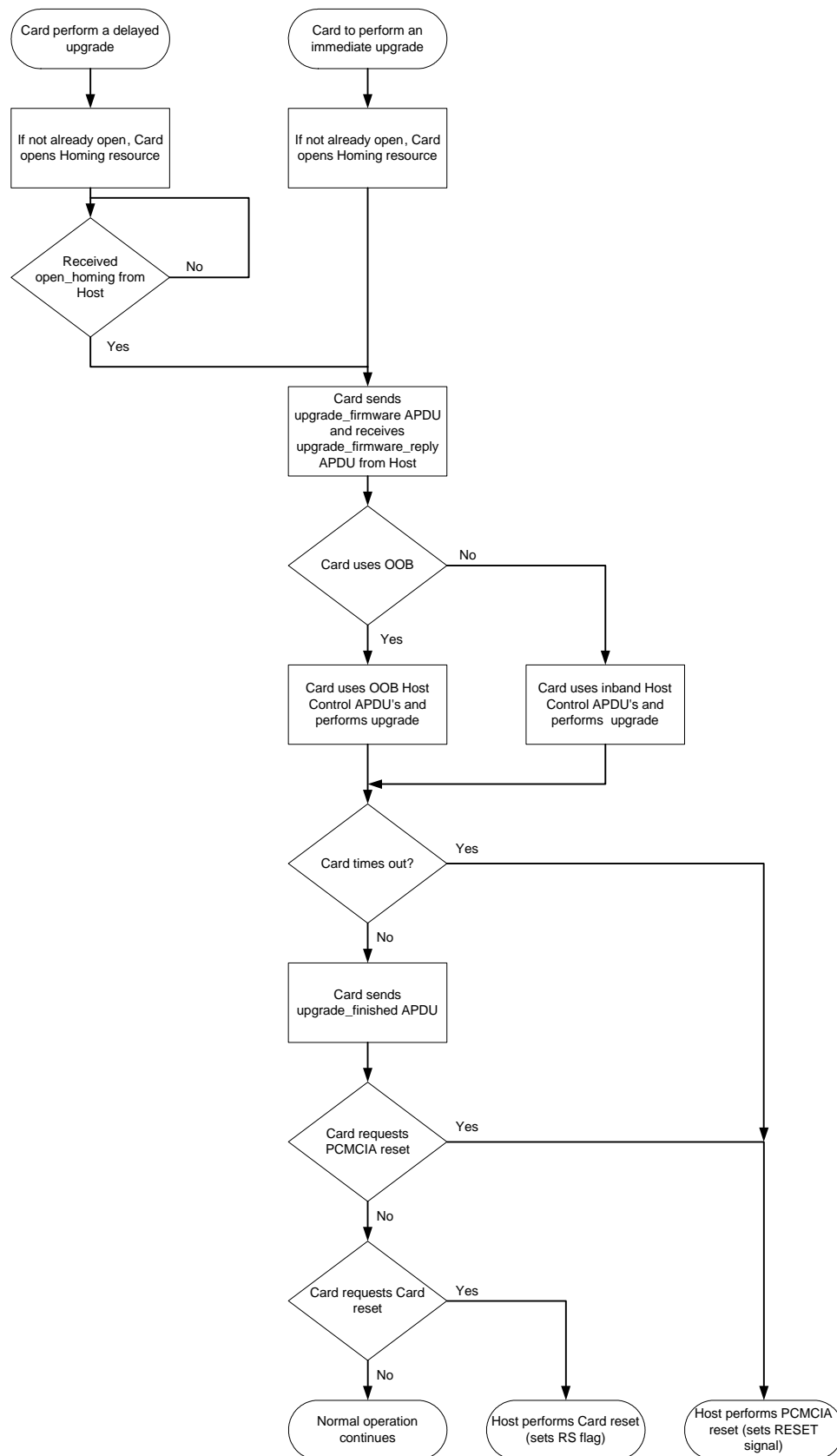


Figure 9.18-1 - Firmware Upgrade Flowchart

## 9.18.4 Homing Resource

### 9.18.4.1 Homing Resource Definition

As defined in section 8.8.1.1 of [NRSSB], the Homing resource allows for the Card to request specific services from the Host when the Host is in a standby state. When the Host is in a standby state, only the “immediate” modes will be supported. This resource SHALL be modified to the following definition.

**Table 9.18–1 - Homing Resource**

Resource	Mode	Class	Type	Version	Identifier (hex)
Homing	S-Mode/M-Mode	17	1	2	0x00110042

The Card will open the Homing resource when it requires a firmware upgrade or requires a service. The creation of Homing resource session includes the following objects:

**Table 9.18–2 - Homing Objects**

Apdu_tag	Tag value	Resource	Direction Host ↔ Card
open_homing	0x9F9990	Homing	→
homing_cancelled	0x9F9991	Homing	→
open_homing_reply	0x9F9992	Homing	←
homing_active	0x9F9993	Homing	→
homing_complete	0x9F9994	Homing	←
firmware_upgrade	0x9F9995	Homing	←
firmware_upgrade_reply	0x9F9996	Homing	→
firmware_upgrade_complete	0x9F9997	Homing	←

### 9.18.4.2 open\_homing

The *open\_homing()* APDU is transmitted by the Host to the Card when it enters the standby state, either from power up or from user action. It SHALL send this independent of whether the Host Control resource has a session active.

**Table 9.18–3 - Open Homing Object Syntax**

Syntax	# of bits	Mnemonic
<pre>open_homing() {     open_homing_tag     length_field() }</pre>	24	uimsbf

**open\_homing\_tag**      0x9F9990

### 9.18.4.3 open\_homing\_reply ()

The *open\_homing\_reply()* APDU is transmitted by the Card to the Host to acknowledge receipt of the *open\_homing()* APDU.

**Table 9.18–4 - Open Homing Reply Object Syntax**

Syntax	# of bits	Mnemonic
<pre>open_homing_reply() {   open_homing_reply_tag   length_field() }</pre>	24	uimsbf

**open\_homing\_reply\_tag**      0x9F9992

#### 9.18.4.4 homing\_active

The *homing\_active()* APDU is transmitted by the Host to the Card to inform the Card that the homing request has been activated.

**Table 9.18–5 - Homing Active Object Syntax**

Syntax	# of bits	Mnemonic
<pre>homing_active() {   homing_active_tag   length_field() }</pre>	24	uimsbf

**homing\_active\_tag**      0x9F9993

#### 9.18.4.5 homing\_cancelled

If the Host was not informed that a firmware upgrade was in progress, then it SHALL have the capability to close the homing state.

**Table 9.18–6 - Homing Cancelled Object Syntax**

Syntax	# of bits	Mnemonic
<pre>homing_cancelled() {   homing_cancelled_tag   length_field() }</pre>	24	uimsbf

**homing\_canceled\_tag**      0x9F9991

#### 9.18.4.6 homing\_complete (Normative)

When the Card no longer needs the homing function, then it can transmit a *homing\_complete()* APDU to the Host.

**Table 9.18–7 - Homing Complete Object Syntax**

Syntax	# of bits	Mnemonic
<pre>homing_complete() {   homing_complete_tag   length_field() }</pre>	24	uimsbf

**homing\_complete\_tag**      0x9F9994

#### 9.18.4.7 *firmware\_upgrade*

If the Card uses an in-band channel to perform a firmware upgrade, it SHALL transmit the *firmware\_upgrade()* APDU to the Host. If the upgrade\_source is equal to the QAM inband channel (01), then the Host SHALL immediately give access to the inband tuner through the Host Control resource tune APDU. The Host SHALL NOT interrupt a firmware upgrade until it receives the *firmware\_upgrade\_complete()* APDU. If the Host is not in the standby mode, then it SHALL display the user\_notification\_text as found in [ISO8859-1]. The estimated time to download in download\_time SHALL be in seconds.

**Table 9.18–8 - Firmware Upgrade Object Syntax**

Syntax	# of bits	Mnemonic
firmware_upgrade() {		
firmware_upgrade_tag	24	uimsbf
length_field()		
upgrade_source	8	uimsbf
download_time	16	uimsbf
timeout_type	8	uimsbf
download_timeout_period	16	uimsbf
text_length	8	uimsbf
for(i=0; i<text_length; i++) {		
user_notification_text	8	uimsbf
}		
}		

**firmware\_upgrade\_tag** 0x9F9995

**upgrade\_source** This SHALL define which path the Card will use for its firmware upgrade.

- 0x00 Unknown – Card is not informing Host of source
- 0x01 QAM Inband Channel – Host Control resource will be used
- 0x02 QPSK OOB Channel – Host Control resource will be used
- 0x03 – 0xFF Reserved

**download\_time** The amount of time, in seconds, that it estimated to take for the firmware upgrade. If the value is 0000, then the value is unknown.

**timeout\_type** The type of timeout requested.

- 0x00 Both timeouts – Use both 5 second and download\_timeout\_period
- 0x01 Transport timeout only – 5 second timeout on transport layer
- 0x02 Download timeout only – Value in download\_timeout\_period
- 0x03 No Timeout – Host will not timeout Card
- 0x04 – 0xFF Reserved

**download\_timeout\_period** The amount of time, in seconds, after the Host has received the *firmware\_upgrade()* APDU that the Host should use to determine that the Card has become unstable. After this time, the Host should perform a PCMCIA reset on the Card. The Host's timer should be reset every time a *firmware\_upgrade()* APDU is received. A value of 0000 is defined to be an infinite timeout period.

**user\_notification\_text** The text to be displayed to the user if the Host is not in standby mode.

#### 9.18.4.8 *firmware\_upgrade\_reply*

The Host will reply to the *firmware\_upgrade()* APDU. The Card will not start the download operation until it receives this reply.

**Table 9.18–9 - Firmware Upgrade Reply Object Syntax**

Syntax	# of bits	Mnemonic
<pre>firmware_upgrade_reply() {     firmware_upgrade_reply_tag     length_field() }</pre>	24	uimsbf

**firmware\_upgrade\_reply\_tag** 0x9F9996

#### 9.18.4.9 *firmware\_upgrade\_complete*

After the Card has completed its upgrade, it will transmit the *firmware\_upgrade\_complete()* APDU to the Host. Included in this is whether the Card needs a PCMCIA reset (RESET signal active), Card reset (RS flag active), or no reset. If there is no reset, then the Host may take control of the tuner if the source was inband.

**Table 9.18–10 - Firmware Upgrade Complete Object Syntax**

Syntax	# of bits	Mnemonic
<pre>Firmware_upgrade_complete() {     firmware_upgrade_complete_tag     length_field()     reset_request_status }</pre>	24	uimsbf
	8	uimsbf

**firmware\_upgrade\_complete\_tag** 0x9F9997

#### **reset\_request\_status**

This contains the status of the reset for the Card.

- 0x00 PCMCIA reset requested – The HOST will bring RESET signal active then inactive.
- 0x01 Card reset requested – Host will set RS flag and begin interface initialization
- 0x02 No reset required – Normal Operation continues
- 0x03 0xFF Reserved

Note that if the Card wishes to cancel the firmware upgrade, it can send the *firmware\_upgrade\_complete()* APDU with no reset requested. Normal operation should continue if the Host receives this APDU.

## 9.19 Support for Common Download Specification

The Card SHALL support common download as defined in this document.

This section specifies a common download protocol for CHI for Host devices with OOB data channels.

Annex E of this document provides further clarification for the expected values in the DownloadInfoIndicator Message used with Common Download.

### 9.19.1 Overview of Protocol

The protocol described in this document is based on the DSM-CC data carousel, which provides a format for data objects on a broadcast carousel. Since a common transport layer protocol for the in-band data channel (also known as the Forward Application Transport or FAT Channel) is MPEG-2, it provides a convenient starting point for a common protocol to download operating software code objects.

Control messages, specific to each type of Host device on the network, provide a locator (frequency of the transport stream, modulation mode, and PID) for the code file on the data carousel. This document uses the following terminology:



**OOB Forward Data Channel method:** This method places the Code Version Table (CVT), as defined in section 3.5.5, in the [OCCD]. The Card acquires the CVT via the OOB, filters the CVT and passes relevant information to the Host, as defined within this document. The Host utilizes the information passed to it via the Card to determine if a download is available. Download via this method is not possible without the Card. The data carousel is carried on the IB and contains the code file image. The Host only knows of the existence of a download via the Card.

Currently, the transport and message protocols between the Headend and the Card are proprietary. In order for any Host to decode the message, extensions to the existing CHI Specification, are required. These extensions provide a common network interface to the Host. This approach requires some additional functionality in the Card, in that the Card translates the proprietary network protocols to the common one specified in this document.

There are four types of code upgrade protocols that may be used by the MSO to download code. Additionally, there may be devices on the network for which the MSO does not have code objects, in which case, download would not be supported. Summarizing these options as follows:

- Broadcast
- Command
- On-Demand
- DOCSIS
- Not supported

This specification defines protocols for the first three download methods; the fourth method is well documented in the DOCSIS specification. These methods specified here are based on the DSM-CC data carousel. The broadcast model is used when the MSO is not dynamically adding files to the broadcast carousel. The command model is used when the MSO has just added a new code file and wants all applicable Host devices to download the code file immediately. The on-demand case allows the MSO to dynamically add and remove code images as new subscribers come online and require upgrades.

#### **9.19.1.1 Common Download via the OOB Forward Data Channel**

In the broadcast model, data is broadcast over the OOB Forward Data Channel that relates the manufacturer and hardware version to the locator for the code object in a DSM-CC data carousel. The Card filters these data and passes the appropriate data onto the Host. The Host can then tune to the appropriate broadcast MPEG multiplex stream and set the PID filters to the PID that identifies the code object in the multiplex stream.

Because every possible code object might not be carried all of the time on the broadcast carousel, the MSO may provide an on-demand capability. In this method, when a Host signs onto the network, the Headend is informed that a new Host is now on-line. If a new version of the software for that Host is available, the Headend loads the object onto the carousel and sends a message back to the Host identifying the location of this code object. If a code object is not available, the Host is informed that download is unsupported. After the Host has finished downloading the object and authenticates it, it sends a 'done' message so that the Headend can unload that object from the carousel.

#### **9.19.2 Operational Details**

The Card utilizes the *host\_info\_request()* APDU, as defined in this document, to instruct the Host as to which type of download method to utilize, the OOB Forward Data Channel or DOCSIS. The Host SHALL NOT make any assumptions as to what type of download to utilize. The MSO may opt to initiate a DOCSIS download without informing the Card; this scenario is not within the scope of this document.

### **9.19.2.1 Download Protocols**

#### **9.19.2.1.1 One-way, Broadcast**

The DSM-CC data carousel is placed on a multiplex. The headend inserts a CVT in the OOB. The Card filters all incoming CVTs and determines if a download is applicable for the attached Host. If the download is applicable, then the Card informs the Host of the available download. The Host acquires the DSM-CC data carousel and downloads the code object. If a code file for a particular host is not defined in the CVT, then code download for that host is not supported.

#### **9.19.2.1.2 Two-way, Command**

After the Headend receives a new code file for a Host and/or a set of Hosts that requires an immediate download, the Headend loads the new code file on the IB DSM-CC data carousel, which is placed on a multiplex. The headend inserts a CVT in the OOB. The Card filters all incoming CVTs and determines if a download is applicable for the attached Host. If the download is applicable, then the Card informs the Host of the available download. The Host acquires the DSM-CC data carousel and downloads the code object. The main difference between this download method and the broadcast is that the Host SHALL download as indicated by the download command. If a code file for a particular host is not defined in the CVT, then code download for that host is not supported.

#### **9.19.2.1.3 Two-way, On-Demand**

When a Host receives a CVT and determines that an image download is required, it will either:

- always notify the Card to notify the headend, or
- it will look at the DSM-CC carousel location in the CVT. If it does not detect its image file, it will notify the Card to notify the Headend. After the Headend is notified that a particular Host has requested a download, the Headend loads the appropriate code file onto the in-band broadcast carousel.

#### **9.19.2.1.4 DOCSIS**

This method uses a two-way connection through a DOCSIS cable modem and utilizes the Trivial File Transport Protocol (TFTP) method used by DOCSIS for its operational software. For Host devices (with embedded Cable Modem), in DOCSIS enabled networks, this mechanism can, at the discretion of the operator, be used to upgrade the operating software. Further elaboration of this method is beyond the scope of this specification.

##### **9.19.2.1.4.1 DSG OOB**

The DSM-CC data carousel is placed on a multiplex. The headend inserts a CVT into the DSG server to be placed in the OOB. The Card filters all incoming CVTs on the appropriate DSG tunnel and determines if a download is applicable for the attached Host. If the download is applicable, then the Card informs the Host of the available download. The Host acquires the DSM-CC data carousel and downloads the code object. If a code file for a particular host is not defined in the CVT, then code download for that host is not supported.

#### **9.19.2.1.5 Not Supported**

The MSO reserves the right to not support upgrade of any Host.

### **9.19.2.2 Level of support for OpenCable devices**

OpenCable Bidirectional Single Tuner Hosts SHALL support the broadcast in-band carousel method, the in-band command method, and the on-demand method. The Multi-Tuner Host SHALL also support the DOCSIS method.

### 9.19.2.3 DSM-CC Data Carousel

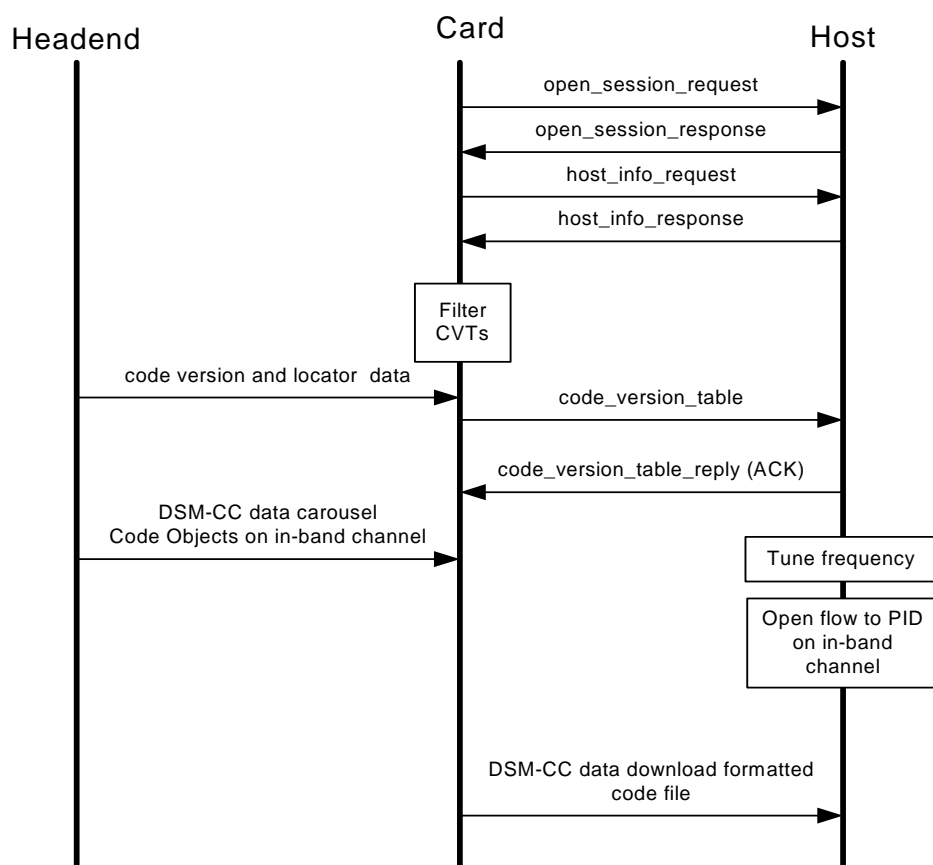
All software objects SHALL be transported over the in-band, broadcast channel via the DSM-CC data carousel [ISO13818-6]. The DSM-CC specification does not require the DSM-CC control messages. The Download Information Indication message, as defined in section 7.3.2 of [ISO13818-6] and modified as defined in Section 3.2.1 of [OCCD], and the message sequence for data carousel scenario, as defined in section 7.5 of [ISO13818-6] SHALL be supported. The CableCARD-Host control messages are defined in this specification.

### 9.19.2.4 Download Operation

The download method used by the operator is optional and in part depends upon the capabilities of the network and the Hosts on the network.

#### 9.19.2.4.1 One-Way Operation

The following figure describes the communication between the Headend and the Card, and the Card and the Host.



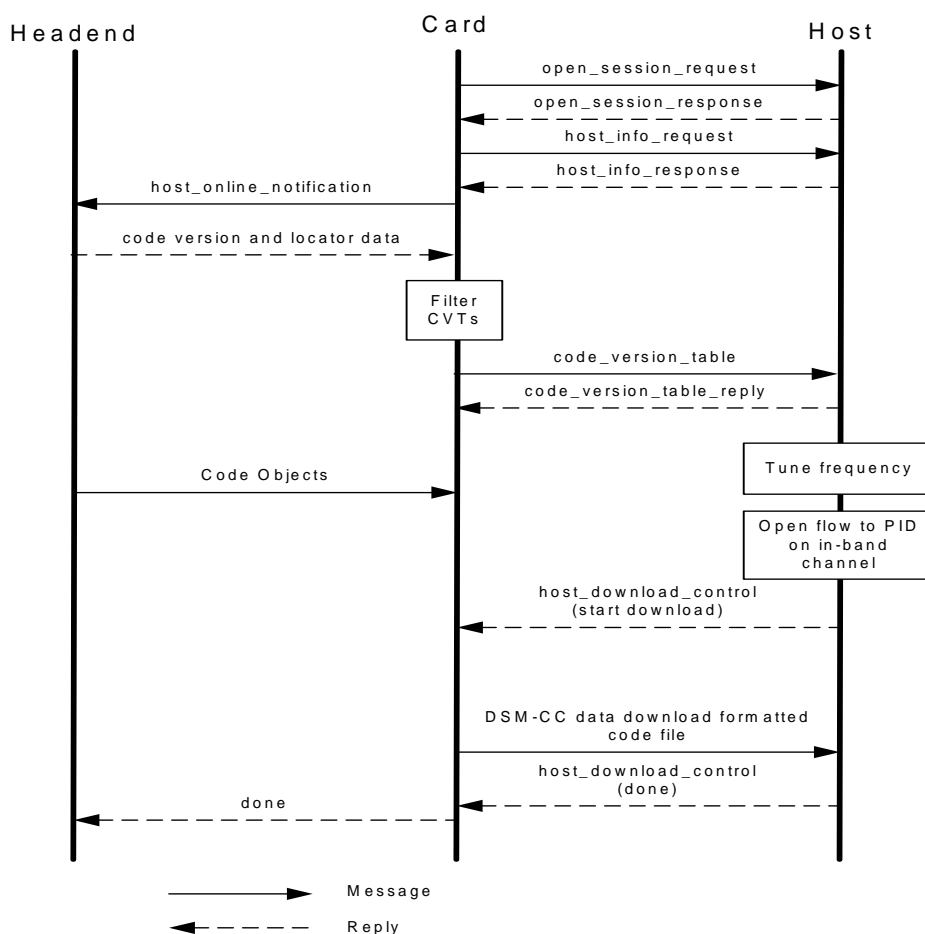
**Figure 9.19-1 - One-Way Operation**

After a session is opened between the Card and the Host, the Card requests identification information from the Host. The Host responds with the `vendor_id` (OUI), `hardware_version_id` from the Host (`host_info_response`). The Card uses this data to filter the Code Version Tables (CVT) that are broadcast over the OOB channel. Each CVT corresponds to a different hardware and software version. The locator is contained within the CVT and is the frequency of the transport stream and the PID for the specific data carousel of the code file. The Card transmits the proper CVT to the host. The Host determines if a download is required by comparing the code file name in the CVT to that store in the Host. If a download is required, the Host begins to download the code object by tuning to the proper in-band frequency and selecting the proper PID in the in-band multiplex stream.

### 9.19.2.4.2 Two-Way Operation – Command

The two-way operation is similar to the one-way operation, except that there is some handshaking between the Card and the Headend. After the Host receives a CVT and determines that the download is commanded, the Host initiates the download as indicated in the command type field of the CVT.

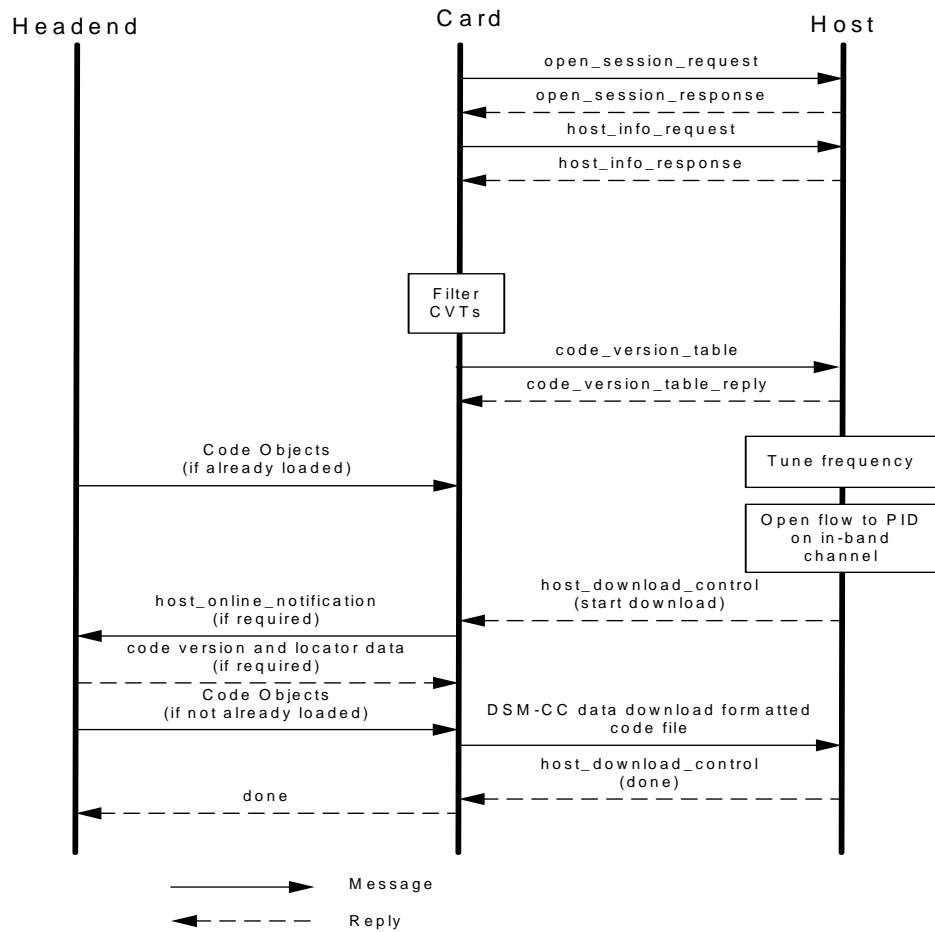
The following diagram describes the communication between the Headend and the Card and the Card and the Host.



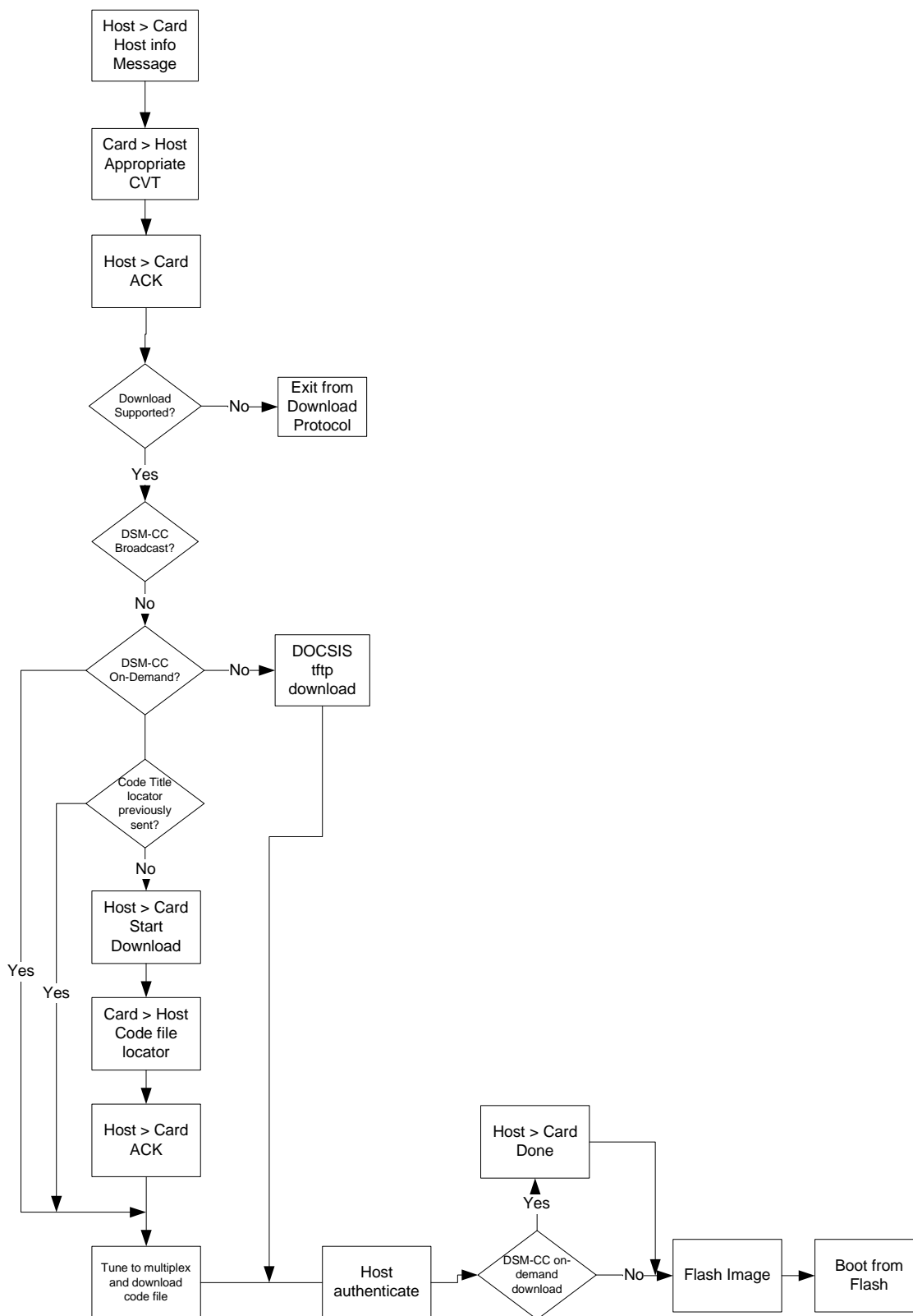
**Figure 9.19-2 - Two-Way Operation– Command**

### 9.19.2.4.3 Two-way Operation- On-Demand

The two-way operation is similar to the one-way operation, except that there is some handshaking between the Card and the Headend. After the Host receives a CVT and determines that a new code image exists, it is told in the CVT to either notify the Card to inform the Headend of this hardware or to first determine if its code image is already loaded on the DSM-CC carousel and only notify the Card to inform the Headend of the hardware if it is not. If download is not supported, the default CVT is sent (“Download Unsupported”, download\_type equals 03, see Section 9.19.3.5).



**Figure 9.19-3 - Two-Way Operation – On-Demand**

**9.19.2.5 Operational Summary****Figure 9.19-4 - Flow Chart Summarizing Download Operations**

### 9.19.2.6 Code Authentication

After a code image is downloaded into the set top box and before it is placed in permanent storage in non-volatile memory, the image is authenticated using the [SCTE23] code authentication process regardless of the method used to download the file. This method specifies a particular structure to the code file (PKCS#7 compliant). The code file consists of the manufacturer's Code Verification Signature (CVS), and X.509 Code Verification Certificate (CVC) signed by the root CA, and the signed code image that is compatible with the target.

### 9.19.3 System Control Resource

This section provides details of Host-Card messages. A new resource, the System Control resource, is introduced for handling revision control and download operations. Applications SHALL exist in the Card to support this resource. New Application Protocol Data Units (APDU) are also introduced.

#### 9.19.3.1 Resource Identifier

The following resource identifier associated with the System Control resource SHALL reside in the Host and is optional for use by the Card. The Card SHALL open a session to this resource in the Host and SHALL NOT close it. Only one session is supported by the Host.

**Table 9.19–1 - Resource Identifier**

Resource	Mode	Class	Type	Version	Identifier (hex)
System Control	S-Mode/M-Mode	43	1	1	0x002B0041

#### 9.19.3.2 Application Objects (APDUs)

The following table is a list of the APDUs that are required for this specification. The *host\_info\_request()*, *host\_info\_response()* and the *host\_download\_control()* APDUs are required for both download methods. The *code\_version\_table()* and *code\_version\_table\_reply()* APDUs are exclusively utilized for the OOB Forward Data Channel download method. The *host\_download\_command()* APDU is exclusively utilized for the IB FAT Channel download method.

**Table 9.19–2 - Table of Application Protocol Data Units**

APDU_tag	Tag value (hex)	Resource	Direction Host <-> Card
Host_info_request	9F9C00	System Control	←
Host_info_response	9F9C01	System Control	→
Code_version_table	9F9C02	System Control	←
Code_version_table_reply	9F9C03	System Control	→
Host_download_control	9F9C04	System Control	→
Host_download_command	9F9C05	System Control	←

#### 9.19.3.3 host\_info\_request

After the Card opens a session to the System Control resource, the Card SHALL query the Host to determine its vendor ID and hardware version ID and optional additional parameters. The Card also SHALL inform the Host as to what type of download the Headend is going to use to update the Host. The Card SHALL use at least the vendor ID and hardware version ID to filter the CVT. If a download is in progress, the Host SHALL terminate it.

**Table 9.19–3 - host\_info\_request**

Syntax	# of bits	Mnemonic
host_info_request() { host_info_request_tag length_field() supported_download_type }	24	uimsbf
	8	uimsbf

**host\_info\_request\_tag** Value = 0x9F9C00

**supported\_download\_type** Defines the type of Common Download method utilized by the Headend.  
0x00 OOB Forward Data Channel method  
0x01 Reserved  
0x02 DOCSIS only  
0x03 – 0x FF Reserved

Note: this document does not define DOCSIS download requirements. A headend need not use this ADPU to inform the Host that updates are performed via a DOCSIS download.

#### 9.19.3.4 host\_info\_response

The Host SHALL respond to the Card query with its vendor ID and hardware version ID.

**Table 9.19–4 - host\_info\_response**

Syntax	# of bits	Mnemonic
host_info_response() { host_info_response_tag length_field() vendor_id hardware_version_id number_of_descriptors for (I=0; i<number_of_descriptors; i++) { descriptor_tag descriptor_len descriptor_data() } }	24	uimsbf
	24	uimsbf
	32	uimsbf
	8	uimsbf
	8	uimsbf
	8	uimsbf

**host\_info\_response\_tag** Value = 0x9F9C01

**vendor\_id** Organizationally Unique Identifier (OUI) assigned to the Host device vendor by the IEEE. A value of 0x000000 is not valid.

**hardware\_version\_id** Unique Hardware identifier assigned to each type of hardware from a particular vendor. The hardware\_version\_id assigned by the vendor SHALL be a unique number (for a given vendor\_id) and correspond on a one-to-one basis with the Model ID of the Host. The hardware\_version\_id SHALL be stored in the Host and reported to the Card (S-Mode and M-Mode) in a robust manner.

The same Model ID SHALL correspond on a one-to-one basis with the unique hardware\_version\_id number required under this specification.

**number\_of\_descriptors** Indicates the number of descriptors defined in the following fields.

**descriptor\_tag** Possible Values:



- 0 descriptor\_data is Host proprietary data. The maximum value for descriptor\_len is 128.
- 1-127 reserved for future standardization
- 128-255 optional, for use by Card-Host pairs, where both Card and Host support the same implementation of the Specific Application Resource. Other Card -Host pairs SHALL skip these descriptors using descriptor\_len value.

### 9.19.3.5 code\_version\_table

The Headend broadcasts all CVTs via the OOB-FDC. After the Card receives the *host\_info\_response()* message from the Host, the Card SHALL only then start filtering any CVTs it receives from the Headend and passing them to the Host. The Card SHALL pass a CVT to the Host only if it meets all of the following criteria:

- CVT vendor\_id matches Host vendor\_id, and
- CVT hardware\_version\_id matches Host hardware\_version\_id

Only one code object SHALL be on the carousel at any given time for a given vendor\_id and hardware\_id.

The Host acknowledges the receipt of the CVT and responds with an OK or an appropriate error code message in the code\_version\_table\_reply. The Card continues to transmit the CVT until it receives the ACK message. If a new, different CVT is received during this time, the Card SHALL transmit it to the Host, if appropriate to the selection criteria described above.

It is up to the Host to determine if a download is required, the Card SHALL NOT determine this. When the Host receives a valid CVT (the OUI and hardware\_version\_id match), the Host SHALL determine if the code file name in the CVT matches the code file name stored in non-volatile memory when the code was last updated. If the file names do not match, the action of the Host SHALL be according to the download\_command parameter. If the length of the CVT software upgrade filename is different than the length of the Host software upgrade filename, then they SHALL be declared to be different, independent of their contents. If the download\_command parameter equals 03 (download now, no exceptions), then the Host SHALL ignore the code file name that is stored in non-volatile memory and SHALL begin to download of the file referred to in the CVT.

It is also up to the Host design to handle error conditions without lockouts or wait states as well as to authenticate the vendor parameters and download code. The Host SHALL assume that the Card is operating correctly.

The use of frequency vector and PID avoids the use of the virtual channel table, which assumes that the entire SI is being processed. When the download is on-demand (download\_type parameter equal 01), the Card MAY send a CVT with frequency and PID equal to 0, to signify that the location of the code file is not known. In this case, an additional CVT with the frequency and PID should be sent when the location is known.

In a program corruption case, the CVT may not be available.

**Table 9.19–5 - Code Version Table**

Syntax	# of bits	Mnemonic
code_version_table() {		
code_version_table_tag	24	uimsbf
length_field()		
number of descriptors	8	uimsbf
for(i=0;i<number of descriptors;i++){		
descriptor_tag	8	uimsbf
descriptor_len	8	uimsbf
descriptor_data()		
}		
download_type	4	uimsbf
download_command	4	uimsbf
frequency_vector	16	uimsbf
transport_value	8	uimsbf
Reserved	3	uimsbf
PID	13	uimsbf
code_file_name_length	8	uimsbf
for(i=0;i<software_filename_length;i++){		
code_file_name_byte	8	uimsbf
}		
code_verification_certificate()		
}		

**download\_status\_tag** Value = 0x9F9C02

**number\_of\_descriptors** SHALL be greater than 2, mandatory descriptors are vendor\_id and hardware\_version\_id

**descriptor\_tag** Possible Values:

- 0 descriptor\_data is vendor\_id (mandatory, descriptor\_len = 3). Unique Identifier (the vendor's OUID) assigned to each vendor.
- 1 descriptor\_data is hardware\_version\_id (mandatory, descriptor\_len = 4), Unique Hardware identifier assigned to each type of hardware from a particular vendor.
- 2 descriptor\_data is Host proprietary data. The maximum value for descriptor\_len is 128.

reserved for future standardization

128-255 optional, for use by Card-Host pairs, where both Card and Host support the same implementation of the Specific Application Resource. Other Card-Host pairs SHALL skip these descriptors using descriptor\_len value.

**download\_type** Type of download (supplied by Headend):

- 0x00 One-Way, broadcast
- 0x01 Always on Demand
- 0x02 DOCSIS tftp
- 0x03 Download unsupported. No code Object available. In this case, the code\_file\_name\_length, frequency\_vector and PID parameters SHALL be equal to zero. Note: If the Host does not receive a CVT, then it SHALL assume that no code object is available.
- 0x04 Conditional On Demand

**download\_command** Possible Values:

- 0x00 Download Now
- 0x01 Deferred Download
- 0x02 Download now, no exceptions
- 0x03 reserved

<b>frequency_vector</b>	Frequency of the download carousel. The frequency is coded as the number of 0.25 MHz intervals. If download_type parameter equals 02, this parameter SHALL be set to 0
<b>transport_value</b>	Possible Values: 0x00 DOCSIS Channel 0x01 FAT Channel/QAM64 0x02 FAT Channel/QAM256 0x03 – 0xFF Reserved
<b>PID</b>	Stream identifier of the code file. If download_type parameter equals 02, this parameter SHALL be set to 0.
<b>code_file_name_length</b>	Length of code file name
<b>code_file_name_byte</b>	Name of software upgrade file on carousel. This is the name of the Code File (see [SCTE23]) that is on the broadcast carousel as well as in Host Flash. For download_type = 0x01, 0x02, and 0x04, the DSM-CC data carousel SHALL carry the Code File Name in the Download Info Indication message, module_info_byte loop. All bytes in the <i>code_version_table()</i> APDU code_file_name_byte loop and the associated byte in the Download Info Indication message module Info Byte loop SHALL be the same. The Download Info Indication compatibility Descriptor SHALL be ignored by the Host when using the OOB forward download method.”
<b>code_verification_certificate</b>	Authentication certificate(s) per [SCTE23]

### 9.19.3.6 code\_version\_table\_reply

When the Host receives a *code\_version\_table()* APDU, it SHALL respond with the *code\_version\_table\_reply()* APDU. This response serves as an acknowledgement to the receipt of the CVT and an error code if necessary. When the Card receives this APDU, it SHALL stop sending CVTs to the Host.

**Table 9.19–6 - code\_version\_table\_reply**

Syntax	# of bits	Mnemonic
code_version_table_reply() { code_version_table_reply_tag length_field() host_response }	24	uimbsbf
	8	uimbsbf

**code\_version\_table\_reply\_tag** Value = 0x9F9C03

**host\_response** Host response to download status:  
0x00 ACK, no error  
0x01 Invalid vendor ID or hardware version ID  
0x02 Other parameter error  
0x03-0xFF Reserved

### 9.19.3.7 host\_download\_control

This APDU is used when the download\_type parameter equals 01 (on-demand download) in CVT.

If the Host was told to do conditional\_on\_demand, the Host SHALL first determine if its new download image is already loaded on the DSM-CC carousel. If it is, then it SHALL send the host\_download\_control with the host\_command as (00) start download.

If the Host was told to do conditional\_on\_demand and it does not find its download image, or it was told to do always\_on\_demand, then it SHALL send the host\_download\_control with the host\_command as (02),

notify\_headend. The Card SHALL then send a host\_notification to the Headend along with the vendor\_id and hardware\_version\_id. The Headend SHALL then send the host\_download\_control with the host\_command as (00) start download.

When the Host has successfully authenticated the code file and has requested that the Card send the host\_notification, it SHALL send an additional *host\_download\_control()* APDU with a host\_command parameter equal to 01 (done). The Card SHALL then send the done message to the Headend along with the vendor\_id and hardware\_version\_id so that the code file can be unloaded from the carousel.

**Table 9.19–7 - host\_download\_control table**

Syntax	# of bits	Mnemonic
host_download_control() { host_download_control_tag length_field() host_command }	24	uimsbf
	8	uimsbf

**host\_download\_control\_tag** Value = 0x9F9C04

**host\_command** Host command:

- 0x00 Start download
- 0x01 Download Completed – sent when done receiving data.
- 0x02 Notify headend
- 0x03-0xFF Reserved

### 9.19.3.8 host\_download\_command

The Card SHALL utilize the *host\_download\_command()* APDU to command a Host to initiate a download when using the two-way Inband FAT channel commanded download method. The Card SHALL also utilize this APDU to command a Host to use the values defined within the APDU to locate CVDTs instead of the source ID when using the one-way Inband Forward Application Transport Channel broadcast download method.

**Table 9.19–8 - host\_download\_command**

Syntax	# of bits	Mnemonic
host_download_command() { host_download_control_tag length_field() host_command location_type if(location_type == 00){ source_id } if(location_type == 01){ frequency_vector transport_value stream_ID if(stream_ID == 00){ Reserved PID } if(stream_ID == 01){ program_number } } }	24   8 8 16  16 8 8 3 13  16	uimsbf   uimsbf uimsbf uimsbf  uimsbf uimsbf uimsbf uimsbf uimsbf  uimsbf

<b>host_download_control_tag</b>	Value = 0x9F9C05
<b>host_command</b>	<p>Defines the priority of download.</p> <p>0x00 Check for download during next cycle using source_id</p> <p>0x01 Download now</p> <p>0x02 Deferred download</p> <p>0x03 Download now, no exceptions</p> <p>0x04- 0xFF Reserved</p>
<b>location_type</b>	<p>Defines the method in which the Host device is to utilize to acquire the DSM-CC stream.</p> <p>0x00 Transport Stream location is defined in the channel map and may be found via the defined source ID</p> <p>0x01 Indicates that the Host device is to use the frequency, modulation type and PID or program number to acquire the DSM-CC stream.</p> <p>0x02 – 0xFF Reserved</p>
<b>source_id</b>	The VCT source ID that is associated with each program source. The source ID is utilized to locate the frequency that the DSC-CC data carousel is multiplexed on. A value of zero indicates that the download is located via the previously assigned source ID.
<b>frequency_vector</b>	Frequency of the download carousel. The frequency is coded as the number of 0.25 MHz intervals. If download_type parameter equals 02, this parameter SHALL be set to 0.
<b>transport_value</b>	<p>Defined values as follows:</p> <p>0x00 DOCSIS channel</p> <p>0x01 FAT channel/QAM64</p> <p>0x02 FAT channel/QAM256</p> <p>0x03- 0xFF Reserved</p>
<b>stream_ID</b>	<p>Defines the way in which the DSM-CC is to be located.</p> <p>0x00 Indicates that the DSM-CC stream is to be located utilizing the defined PID.</p> <p>0x01 Indicates that the stream is to be located utilizing the program number.</p> <p>0x02 – 0xFF Reserved</p>
<b>PID</b>	Packet identifier of the stream that contains the code file.
<b>program_number</b>	Defines the program number in which the DSM-CC stream resides.

## 10 EXTENDED CHANNEL OPERATION

The extended channel provides a data path between the Card and the Host. The QPSK modem (the traditional “out-of-band channel”) is one such path. An eCM, when present, provides another. It is implemented as a separate endpoint from the command channel.

### 10.1 Internet Protocol Flows

The Extended Channel supports delivery of IP packets across the Card interface. Both unicast (point-to-point) and multicast (point-to-multipoint) addressing are supported by this protocol. If the Host is in OOB mode, then the Card is expected to service the IP flow via utilization of the Host’s RDC and supply the Host with an IP address. On request of a “new flow request” from the Host, the Card will respond to the request to open the flow by obtaining an IP address for use by the Host. That IP address is returned in the “new flow confirmation” message.

*Informative Note:* The Card is not required to grant a request for service type IP Unicast when requested by the Host.

In DSG mode, the Card resides at the Network Layer and the Host will utilize its eCM to provide the Data Link Layer to the underlying DOCSIS network. When the Card wishes to utilize the DOCSIS network to transfer IP datagrams upstream, it must first submit a “new flow request” to the Host to establish an IP flow to transfer datagrams between the Card and the Host’s eCM interface. The Card will submit its MAC address in its request to the Host for an IP flow.

If the Host grants the new IP flow request, then the Host utilizes DHCP to acquire an IP address for the Card, and sends this information, along with the DOCSIS maximum transmission unit (MTU) (1500 bytes for IP datagrams) to the Card in a new flow confirmation. The Host now opens an IP flow to the Card over the Extended Data Channel.

The Host utilizes the MAC address provided in the Card’s IP flow request to filter Ethernet frames from the eCM that are intended for the Card. The Host extracts all unicast IP datagrams from Ethernet frames addressed to the Card’s MAC address and forwards them over the Extended Channel to the Card.

The Host utilizes the Extended Channel’s IP flow to forward IP datagrams it receives over the eCM interface on behalf of the Card. The Host does not forward to the Card any datagrams received over other interfaces (e.g., Ethernet port, USB port, etc.).

The Host forwards all IP datagrams received from the Card to the eCM interface. The Host does not forward any IP datagrams received from the Card to any other interface, including but not limited to: IEEE-1394, Ethernet, USB, 802.11a/b/g/n/x, Multimedia Over Coax Alliance (MoCA), etc. The Host resolves the destination MAC address of the IP datagrams that it receives from the Card and applies the appropriate MAC addresses to the Ethernet frames it sends upstream.

If an established IP type of flow becomes unavailable for any reason, the device that has granted the flow is required to report that fact to the one that has requested the flow. The “lost flow indication” transaction is used to report this type of event. One example case where a flow may become unavailable is due to a change in the state of the eCM that may have resulted from a change via SNMP to the eCM’s operational state.

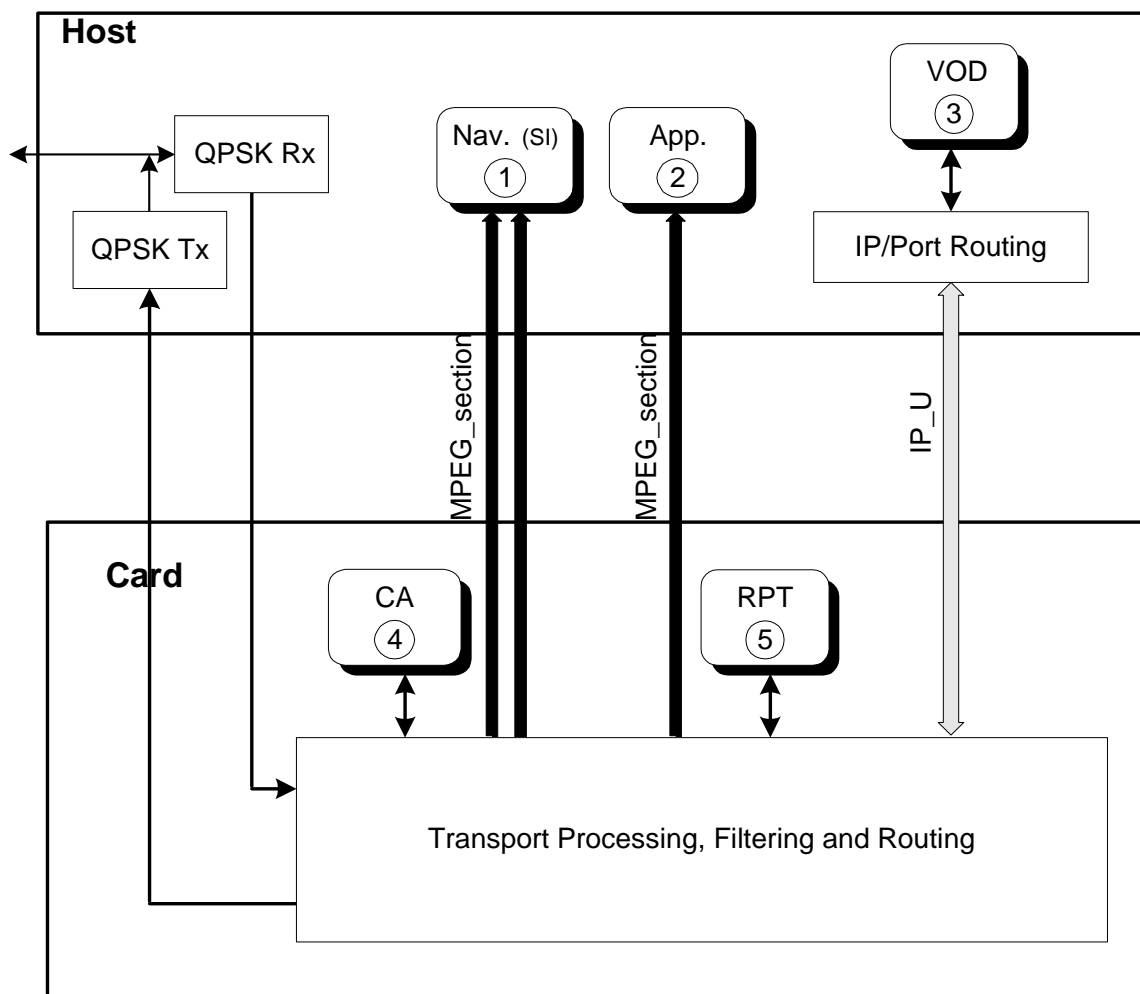
### 10.2 Flow Examples—QPSK Modem Case

Figure 10.2-1 diagrams a CHI in which four flows have been set up. In this example case, the Card provides a full-duplex modem function for the benefit of the Host (as well as itself).

In the figure, the rectangles with rounded corners represent applications. In this example, the Host has a Navigation application that receives Service Information data on the Extended Channel via the Card interface (#1). The Host has opened up three flows to receive MPEG data from the Card, and has supplied different PID values for filtering for each. The navigation function (#1) uses two SI flows in the example, and another application (#2) uses the third flow. The Host may have a Video On Demand (VOD) application (#3).

In Figure 10.2-1, the types of services that the Card is required to support are shown with black arrows. As shown in the figure, three flows delivering MPEG table sections are required. Flows that may be available at the option of

the supplier of the Card are shaded gray. In the figure, the Card supports an IP flow, but a compliant Card can choose not to support the IP service type.



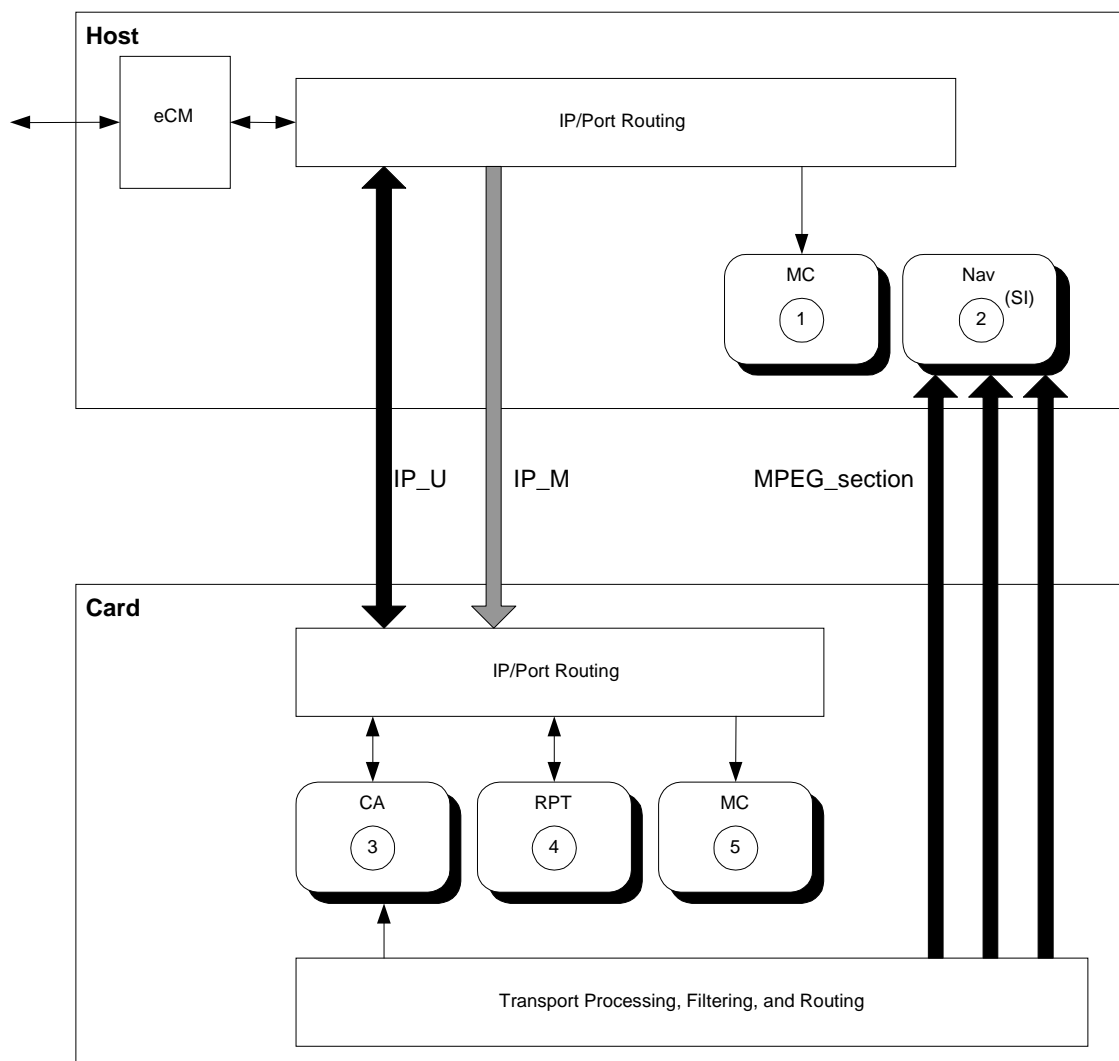
**Figure 10.2-1 - Flow Examples - QPSK Modem Case**

The Card includes two applications of its own. The Conditional Access process (#4) receives data via downstream QPSK. The Card includes a pay-per-view report back function (#5).

Note that none of these Card applications use flows that travel across the Card interface.

### 10.3 Flow Examples— Embedded Cable Modem Case DSG Mode

In the next example case, the Host includes an eCM. Figure 10.3-1 diagrams a CHI in which five flows have been set up. When a Host includes an eCM, the Host is required to support at least one flow of service type IP Unicast (IP\_U). As before, the Card SHALL support three MPEG section flows if the Host requests them.



**Figure 10.3-1 - Flow Examples - eCM Case**

In this example, the Host has some application that uses multicast addressed packets (#1) and a Navigation application (#2) that receives Service Information data on the Extended Channel via the Card interface via three separate flows.

The Navigation application can open three different simultaneous flows, specifying different PID values for each. For example, it might set one to the base PID that carries SI network data including the Master Guide Table, Virtual Channel Table and System Time. It can set a second one to point to a PID value where Event Information Tables for a specific time slot may be found, and another to collect associated Extended Text Tables (ETTs).

The Card includes three applications of its own. The Host routes IP packets to the Card applications based on IP address. For unicast packets, those that match the IP address assigned to the Card will be routed across the interface. For multicast packets, those matching the multicast group address associated with a particular flow will be delivered.

The Card includes a pay-per-view reportback function (#4) that uses standard IP packets for data transport. Finally, the Card includes some application (#5) that has registered with the Host to receive multicast-addressed IP packets through the Host modem.



## 10.4 Summary of Extended Channel Flow Requirement

Compliance with this standard requires Host and Card to support certain flows. Other types of flows may be supported at the discretion of the Host or Card. The following table summarizes the requirements.

**Table 10.4–1 - Flow Requirements**

Requestor	Data Direction	Service Type	Number of Concurrent Flows
Host	Card → Host	MPEG section	6 or more
If Host implements eCM or if OCAP is supported:			
Card	Host ↔ Card	IP Unicast	At least 1
If Host implements DSG:			
Card	Host → Card	DSG	1

## 10.5 System/Service Information Requirements

The Card SHALL supply System and Service Information across the HOST- Card interface, using *Service\_type MPEG\_section*, as defined in Section 9.14.1 and [SCTE65]. The set of MPEG-2 tables provided to support the navigation function in the Host device SHALL conform to one or more of the profiles specified in [SCTE65].

Informative note: Profiles 1 through 5 are compatible with Host devices deployed as of Jan 1, 2000. Host devices that are intended to be portable across the United States will need to function with any of the six profiles of [SCTE65]. For operational considerations of various profiles, see section A.3 in [SCTE65].

## 10.6 Link Layer

The link layer of the Extended Channel fragments the datagram PDU, if necessary, over the limited buffer size of the physical layer, and reassembles the received fragments.

### 10.6.1 S-Mode

The link header includes two control bits and the flow\_id value that has been negotiated by the link device for the application (see Section 9.14), to identify the end-to-end communication flow.

**Table 10.6–1 - S-Mode Extended Channel Link Layer Packet**

Bit							
7	6	5	4	3	2	1	0
L	F	0x00					
flow_id (MSB)							
flow_id							
flow_id (LSB)							
datagram PDU fragment							

**L** Last indicator: if this bit is set to '0', then at least one more datagram fragment follows. If this bit is set to '1', this fragment is the last in the datagram.

**F** First fragment indicator: if this bit is set to '1', then this fragment is the first of the datagram. If this bit is set to '0', this fragment is not the first.

**flow\_id** The 3-byte flow identifier associates the data with a registered flow. The flow\_id is assigned as defined in Section 9.14. The flow\_id value of zero is reserved and is not to be assigned.

### 10.6.2 M-Mode

The link layer of the Extended Channel fragments the datagram PDU, if necessary, over the limited buffer size of the physical layer, and reassembles the received fragments.

The link header contains the flow\_id value that has been negotiated by the link device for the application, see Section 9.14 to identify the end-to-end communication flow.

**Table 10.6–2 - M-Mode Extended Channel Link Layer Packet**

Bit							
7	6	5	4	3	2	1	0
0x00							
flow_id (MSB)							
flow_id							
flow_id (LSB)							
Datagram PDU fragment							

**flow\_id** The 3-byte flow identifier associates the data with a registered flow. The flow\_id is assigned as defined in Section 9.14. The flow\_id value of zero is reserved and is not to be assigned.

For data flows made available to the Host by the Card, the Card is responsible for link layer processing of messages to be transferred across the Extended Channel. It is the Host's responsibility to reassemble the received datagram PDU fragments, and to segment PDUs for delivery across the interface. For data flows made available to the Card by the Host, the roles are reversed.

Received datagram PDU fragments SHALL be reassembled into IP packets, or MPEG-2 table sections, or DSG messages, depending upon the service\_type associated with the flow given by flow\_id. The maximum size of the reassembled PDU (IP packet or MPEG-2 table section or DSG message) SHALL be 4,096 for any Service Type.

### 10.6.3 Maximum PDUs

Datagram PDUs to be transmitted upstream SHALL be segmented into fragments not exceeding the negotiated buffer size. The maximum size of any PDU before fragmentation SHALL be 4,096 bytes for downstream data for any Service Type. The maximum size of any PDU before fragmentation SHALL be 1,500 bytes for upstream data for any Service Type.

## 10.7 Modem Models

There are 3 different network connection models that a Host MAY have:

- Unidirectional (no modem)
- Bidirectional, modem function in the Card
- Bidirectional, modem function in the Host

### 10.7.1 Unidirectional Host Model

For the unidirectional Host model, there is no IP connectivity. The extended channel will be utilized solely for receiving the OOB SI data.

For this model, the Card will be the link device for the OOB SI MPEG data flow.

### 10.7.2 Bidirectional With Modem in Card

For the bidirectional Host model with the modem functionality in the Card, the [SCTE55-2] and the [SCTE55-1] PHY (RF processing, QPSK demodulation and modulation) layer is implemented in the Host, and the Data-link and MAC protocols are implemented in the Card. The details of the OOB hardware implementation are covered in Section 5.10.1 of this specification.

For this model, the Card will be the link device for all flows.

### 10.7.3 Bidirectional With Modem in Host

When the Host implements the DSG functionality, all of the DOCSIS cable modem functionality are implemented in the Host. The OOB data flows are transmitted, utilizing DSG tunneling [DSG]. The Host SHALL be capable of receiving the DSG OOB data flows even if it is unable to connect in two-way mode.

For this model, the Card will be the link device for the OOB MPEG SI data flow, and the Host will be link device for IP data flows and for the DSG flow to the Card/DSGCC.

## 10.8 SI Requirements

OOB SI data is transmitted from the Card to the Host using the protocols defined in [SCTE65]. The Card SHALL always be the source of all OOB SI data to the Host. The Host SHALL NOT use OOB SI data received by any other method. This is because the Card MAY reformat the OOB SI data to meet the requirements of [SCTE65], as there is no requirement for the cable system to transmit the OOB SI data in that format.

A Host MAY use data received on OOB data streams that is outside of the [SCTE65] protocols.

## 10.9 EAS Requirements

The Card MAY receive Emergency Alert messaging on either the FAT channels or the Out-of-Band Forward Data channel (OOB FDC). The EAS message syntax is compatible with MPEG-2 transport and is defined in [J042]. For FAT channel transmission, the EAS message SHALL appear in transport packets with the same PID as those used for Service/System Information (SI) and SHALL be transmitted by the Card to the HOST. The table ID for the EAS message is 0xD8 as defined in [J042]. For out-of-band (OOB) transmission, EAS messages SHALL be processed by the Card and SHALL be transmitted over the Extended Channel according to [J042].

EAS messages are transmitted from the Card to the Host using the protocol defined in [J042]. When a Card is installed in the Host, the Card SHALL always be the source of these messages. The Host SHALL NOT use OOB EAS messages received by any other method. The Card MAY reformat the OOB EAS message to meet the requirements of [J042], as there is no requirement for the cable system to transmit the OOB EAS message in that format.

Note: EAS operation for when a Host does not have a Card installed is outside the scope of this specification.

## 10.10 XAIT Requirements

The Card SHALL forward all received XAIT messaging across the extended channel as defined in the OpenCable Application Platform Specification, OCAP 1.0 document [OCAP].

## Annex A Baseline HTML Profile Support

This annex describes HTML keywords that SHALL be supported by the Baseline HTML Profile and gives requirements for each keyword foreseen on the Host.

The Baseline HTML Profile only supports formatted text messages, in the form of HTML pages, with one hyperlink.

The Application Information resource MAY identify Hosts that support more elaborate HTML pages with multiple hyperlinks and multiple levels of text rendering and graphic support. In such a case, the Card can supply HTML pages that take advantage of these enhanced features.

Note: This extended mode of operation is not described in this annex.

### A.1 Format

#### A.1.1 Display

The Baseline HTML Profile pages SHALL be designed to fit in a 4/3 and 16/9 NTSC display size using the smallest common screen (640 x 480) without vertical and horizontal scrolling.

MMI messages from the Card will be limited to a maximum of 16 lines of 32 characters each. If the MMI message is longer than 16 lines, the message will include up to 16 lines of text plus a hyperlink pointing to an additional page.

All text on every page must be visible on the screen.

The Host device may use screen space below the MMI message for navigation buttons such as “Press MENU to Exit” and/or status information. Host-added navigation buttons and status information, if added, must not obscure any MMI text.

If the HTML from the Card contains a hyperlink, the Host MUST provide instructions on how to navigate to any links contained in the Card’s HTML message. Host-added navigation buttons, if added, must not obscure any MMI text.

The Baseline HTML Profile requires that MMI windows be opaque.

#### A.1.2 Font

The Baseline HTML Profile font SHALL support a minimum of 32 characters per line, and a minimum of 16 lines of characters.

#### A.1.3 Text and Background Color

Under the Baseline HTML Profile, the Host MAY render text color as requested in the HTML data from the Card.

Under the Baseline HTML Profile, the Host MAY render the background color as requested in the HTML data from the Card.

If the HTML data does not include a background and/or text color command, or the Host does not support the background and/or color command, the Host SHALL use either

- black (#000000) text on a light gray (#C0C0C0) background or
- white (#FFFFFF) text on a black (#000000) background.

If the Host device supports either the background color or text color command then it SHALL support both of the commands. It should not support only one of the commands. (Footnote: Supporting only one of the commands could lead to unreadable messages, for example if the Card requests blue text on a white background and the Host supports the text color command but uses the default background color, the result would be blue text on a blue background).

#### A.1.4 Unvisited Link Color

Under the Baseline HTML Profile, the Host MAY render the unvisited link color as requested in the HTML data from the Card. If the HTML data does not include an unvisited link color command, or the Host does not support the unvisited link color command, the Host SHALL use blue (#0000FF).

#### A.1.5 Paragraph

Under the Baseline HTML Profile, the Host MAY align paragraphs as requested by the HTML data from the Card. If the HTML data does not include a paragraph alignment command, or the Host does not support the paragraph alignment command, the Host SHALL use a LEFT paragraph alignment.

#### A.1.6 Image

The Baseline HTML Profile does not include support for images.

#### A.1.7 Table

The Baseline HTML Profile does not include support for tables.

#### A.1.8 Forms

The Baseline HTML Profile doesn't include support for forms.

### A.2 Supported User Interactions

#### A.2.1 Navigation and Links

The Baseline HTML Profile does not define how a hyperlink is navigated and selected. It is up to the Host manufacturer to provide some navigation/selection mechanism to identify the user intention and forward the selected link to the Card using the *server\_query()* APDU. It is up to the Card manufacturer to determine how results are returned to the Card through the URL of the *server\_query()* APDU. The Host SHALL provide a method of user navigation to the hyperlink in the MMI message if one is present.

#### A.2.2 HTML Keywords

Table A–1 lists HTML keywords used in the Baseline HTML Profile (R=Required, O=Optional).

A keyword or a parameter marked as optional MAY be inserted in an HTML page, but MAY not be used by the Host. It SHALL NOT change what is displayed on the screen but only the way of displaying it (basically, it applies to the style).

**Table A–1 - HTML Keyword List**

	Required or Optional
Structure	
<HTML>...</HTML>	R
Begin and end HTML document.	
<BODY>...</BODY>	R
Begin and end of the body of the document, optional attributes of the document	
• bgcolor: background color, default = light gray (#C0C0C0)	O
• text: color of text, default = black (#000000)	O
• link: color of unvisited links, default = blue (#0000FF)	O
<A href= ....> ... </A>	R
Begin and end an anchor.	
• href: URL targeted by this anchor.	R

	Required or Optional
Style Element	
<P>	R
Change of paragraph	
• align: CENTER, LEFT, or RIGHT (default = LEFT)	O
 	R
Force new line.	
<B>...</B> <I> ... </I> <U> ... </U>	O
Character style: bold, italic, and underlined	

### A.3 Characters

An HTML page can refer to all Latin-1 characters by their numeric value by enclosing them between the & and ; symbols. For example, the quotation mark “ can be expressed as &#34; in an HTML page. The characters specified in the Added Latin-1 entity set also have mnemonic names. Thus, the following 3 expressions are interpreted as the character “.

&quot;  
&#34;  
“

*NOTE: Mnemonic expressions are case sensitive.*

Table A–2 defines characters, their numeric and mnemonic expressions that the Baseline HTML viewer SHALL support. Any OpenCable baseline HTML page SHALL NOT use the characters, numeric or mnemonic expressions, which are not defined in Table A–2, the Host MAY ignore the characters which are not defined in Table A–2.

This list is taken from the HTML 4 Character entity references found at:

<http://www.w3.org/TR/REC-html40/sgml/entities.html>

**Table A–2 - Characters**

Character	Name	Numeric Expression	Mnemonic Expression
	Horizontal tab	&#9;	
	Line feed	&#10;	
	Space	&#32	
!	Exclamation mark	&#33;	
"	Quotation mark	&#34;	&quot;
#	Number sign	&#35;	
\$	Dollar sign	&#36;	
%	Percent sign	&#37;	
&	Ampersand	&#38;	&amp;
'	Apostrophe	&#39;	
(	Left parenthesis	&#40;	
)	Right parenthesis	&#41;	
*	Asterisk	&#42;	
+	Plus sign	&#43;	
,	Comma	&#44;	
-	Hyphen	&#45;	
.	Period	&#46;	
/	Solidus (slash)	&#47;	
0		&#48;	

Character	Name	Numeric Expression	Mnemonic Expression
1		&#49;	
2		&#50;	
3		&#51;	
4		&#52;	
5		&#53;	
6		&#54;	
7		&#55;	
8		&#56;	
9		&#57;	
:	Colon	&#58;	
;	Semicolon	&#59;	
<	Less than	&#60;	&lt;
=	Equals sign	&#61;	
>	Greater than	&#62;	&gt;
?	Question mark	&#63;	
@	Commercial at	&#64;	
A		&#65;	
B		&#66;	
C		&#67;	
D		&#68;	
E		&#69;	
F		&#70;	
G		&#71;	
H		&#72;	
I		&#73;	
J		&#74;	
K		&#75;	
L		&#76;	
M		&#77;	
N		&#78;	
O		&#79;	
P		&#80;	
Q		&#81;	
R		&#82;	
S		&#83;	
T		&#84;	
U		&#85;	
V		&#86;	
W		&#87;	
X		&#88;	
Y		&#89;	
Z		&#90;	
[	Left square bracket	&#91;	
\	Reverse solidus	&#92;	
]	Right square bracket	&#93;	
^	Circumflex	&#94;	
_	Horizontal bar	&#95;	
`	Grave accent	&#96;	
a		&#97;	
b		&#98;	
c		&#99;	
d		&#100;	

Character	Name	Numeric Expression	Mnemonic Expression
e		&#101;	
f		&#102;	
g		&#103;	
h		&#104;	
I		&#105;	
j		&#106;	
k		&#107;	
l		&#108;	
m		&#109;	
n		&#110;	
o		&#111;	
p		&#112;	
q		&#113;	
r		&#114;	
s		&#115;	
t		&#116;	
u		&#117;	
v		&#118;	
w		&#119;	
x		&#120;	
y		&#121;	
z		&#122;	
{	Left curly brace	&#123;	
	Vertical bar	&#124;	
}	Right curly brace	&#125;	
~	Tilde	&#126;	
	Non-breaking space	&#160;	&nbsp;
¡	Inverted exclamation	&#161;	&iexcl;
¢	Cent	&#162;	&cent;
£	Pound	&#163;	&pound;
¤	Currency	&#164;	&curren;
¥	Yen	&#165;	&yen;
¦	Broken vertical	&#166;	&brvbar;
§	Section sign	&#167;	&sect;
¨	Umlaut/diaeresis	&#168;	&uml;
©	Copyright	&#169;	&copy;
ª	Feminine	&#170;	&ordf;
«	Left angle quote	&#171;	&laquo;
¬	No sign	&#172;	&not;
-	Hyphen	&#173;	&shy;
®	Reg. trade mark	&#174;	&reg;
—	Macron	&#175;	&macr;
°	Degrees	&#176;	&deg;
±	Plus/Minus	&#177;	&plusmn;
²	Superscript 2	&#178;	&sup2;
³	Superscript 3	&#179;	&sup3;
´	Acute accent	&#180;	&acute;
µ	Micron	&#181;	&micro;
¶	Paragraph sign	&#182;	&para;
·	Middle dot	&#183;	&middot;
¸	Cedilla	&#184;	&cedil;
¹	Superscript 1	&#185;	&sup1;



Character	Name	Numeric Expression	Mnemonic Expression
°	Masculine	&#186;	&ordm;
»	Right angle quote	&#187;	&raquo;
¼	One quarter	&#188;	&frac14;
½	One half	&#189;	&frac12;
¾	Three quarters	&#190;	&frac34;
¿	Inverted question mark	&#191;	&iquest;
À	A Grave	&#192;	&Agrave;
Á	A Acute	&#193;	&Aacute;
Â	A Circumflex	&#194;	&Acirc;
Ã	A Tilde	&#195;	&Atilde;
Ä	A Diaeresis	&#196;	&Auml;
Å	A Ring	&#197;	&Aring;
Æ	AE Diphthong	&#198;	&AElig;
Ç	C Cedilla	&#199;	&Ccedil;
È	E Grave	&#200;	&Egrave;
É	E Acute	&#201;	&Eacute;
Ê	E Circumflex	&#202;	&Ecirc;
Ë	E Diaeresis	&#203;	&Euml;
Ì	I Grave	&#204;	&Igrave;
Í	I Acute	&#205;	&Iacute;
Î	I Circumflex	&#206;	&Icirc;
Ï	I Diaeresis	&#207;	&Iuml;
Ð	Icelandic eth	&#208;	&ETH;
Ñ	N Tilde	&#209;	&Ntilde;
Ò	O Grave	&#210;	&Ograve;
Ó	O Acute	&#211;	&Oacute;
Ô	O Circumflex	&#212;	&Ocirc;
Õ	O Tilde	&#213;	&Otilde;
Ö	O Diaeresis	&#214;	&Ouml;
×	Multiplication	&#215;	&times;
Ø	O Slash	&#216;	&Oslash;
Ù	U Grave	&#217;	&Ugrave;
Ú	U Acute	&#218;	&Uacute;
Û	U Circumflex	&#219;	&Ucirc;
Ü	U Diaeresis	&#220;	&Uuml;
Ý	Y Acute	&#221;	&Yacute;
Þ	Icelandic Thorn	&#222;	&THORN;
ß	Small sharp S	&#223;	&szlig;
à	a Grave	&#224;	&agrave;
á	a Acute	&#225;	&aacute;
â	a Circumflex	&#226;	&acirc;
ã	a Tilde	&#227;	&atilde;
ä	a Diaeresis	&#228;	&auml;
å	a Ring	&#229;	&aring;
æ	ae Diphthong	&#230;	&aelig;
ç	c Cedilla	&#231;	&ccedil;
è	e Grave	&#232;	&egrave;
é	e Acute	&#233;	&eacute;
ê	e Circumflex	&#234;	&ecirc;
ë	e Diaeresis	&#235;	&euml;
ì	i Grave	&#236;	&igrave;
í	i Acute	&#237;	&iacute;

Character	Name	Numeric Expression	Mnemonic Expression
î	i Circumflex	&#238;	&icirc;
ï	i Diaeresis	&#239;	&iuml;
ð	Icenlandic eth	&#240;	&eth;
ñ	n Tilde	&#241;	&ntilde;
ò	o Grave	&#242;	&ograve;
ó	o Acute	&#243;	&oacute;
ô	o Circumflex	&#244;	&ocirc;
õ	o Tilde	&#245;	&otilde;
ö	o Diaeresis	&#246;	&ouml;
÷	Division	&#247;	&divide;
ø	o Slash	&#248;	&oslash;
ù	u Grave	&#249;	&ugrave;
ú	u Acute	&#250;	&uacute;
û	u Circumflex	&#251;	&ucirc;
ü	u Diaeresis	&#252;	&uuml;
ý	y Acute	&#253;	&yacute;
þ	Icenlandic thorn	&#254;	&thorn;
ÿ	y Diaeresis	&#255;	&yuml;

## Annex B Error Handling

Interface errors SHALL be handled as described in Table B–1 below:

**Table B–1 - Error Handling**

	<b>Error Condition</b>	<b>Failure Mechanism</b>	<b>Host Action</b>	<b>SCTE Card Action</b>	<b>Comments</b>
1	Card READY signal does not go active	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.
2	Host reads incorrect CIS values	Card	Host reports error using screen in Figure B–1 - Error Display.	None	Host reports error to user. <sup>1</sup>
3	Host writes incorrect TPCE_IND <sub>X</sub> value to POD configuration register	Host	None	Card cannot perform any action.	Host detects as failure #4 and reports error to user. <sup>1</sup>
4	Host sets command channel RS bit but Card fails to set FR bit within 5-second timeout.	Card	Minimum – Perform 1 PCMCIA rest, Report Error if not successful, Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
5	Host sets extended channel RS bit but Card fails to set FR bit within 5-second timeout.	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.
6	Invalid buffer negotiation - Card data channel (buffer size < 16)	Card	Host either 1) reports error using screen in Figure B-1 - Error Display 2) retry PCMCIA resets up to two times and then report error using screen in Figure B-1 - Error Display, or 3) operate with smaller size	None	Host reports error to user. <sup>1</sup>
7	Invalid buffer negotiation - Host data channel (buffer size < 16 or greater than Card data channel buffer size)	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – CableCARD device works with Host buffer size	Host reports error to user. <sup>1</sup>
8	Invalid buffer negotiation – Card extended channel (buffer size < 16)	Card	Host either 1) reports error using screen in Figure B-1 - Error Display 2) retry PCMCIA resets up to two times and then report error using screen in Figure B-1 - Error Display, or 3) operate with smaller size	None	Host reports error to user. <sup>1</sup>
9	Invalid buffer negotiation – Host extended channel (buffer size < 16 or greater than Card data channel buffer size)	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card works with Host buffer size	Host reports error to user. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
10	Card does not respond to Hosts open transport request within 5 seconds	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.
11	Host does not respond to Card request to open resource manager session within 5 seconds.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
12	Host response to open resource manager session response – resource manager non-existent.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
13	Host response to open resource manager session response – resource manager unavailable.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
14	Host response to open resource manager session response – incorrect version of resource manager.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
15	Host response to open resource manager session response – resource manager busy.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
16	Host response to open resource manager session response – invalid status byte.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
17	Card fails to respond to profile_inq within 5 seconds.	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful. Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.
18	Host resource response – no application information resource.	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>
19	Host resource response – no Host control resource.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Card may not be able to do conditional access properly. NOTE: There is a discussion ongoing about DOCSIS only operation. <sup>1</sup>
20	Host resource response – no system time resource.	Host	None	Minimum – Card continues operation and will not open a session to the system time resource. Preferred – Same as minimum but also reports this in its MMI diagnostics application.	Card operations which require system time will not operate. <sup>1</sup>
21	Host resource response – no MMI resource.	Host	None	Minimum – Card continues operation and will not open a session to the MMI resource.	Card cannot utilize MMI for applications or to report error conditions. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
22	Host resource response – no low speed communications.	Host	None	Minimum – Card continues operation and will not open a session to the low speed communication resource. Preferred – Same as minimum but also reports this in its MMI diagnostic application.	If OOB reverse path not available, then some applications will be unavailable. <sup>1</sup>
23	Host resource response – no homing resource <sup>1</sup>	Host	None	Minimum – Card continues operation and will not open a session to the homing resource. Preferred – Same as minimum but also reports this in its MMI diagnostic application.	Card may have some operational problems (i.e., downloading software). <sup>1</sup>
24	Host resource response – no copy protection resource.	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, it will not open a session to the copy protection resource, reports to headend if possible, reports error to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
25	Host resource response – unknown resource identifier.	Host	None	Minimum – Card continues operation.	Not a failure condition
26	Host fails to respond to open session request within 5 seconds.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
27	Host response to open application info resource session – application info non-existent.	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>
28	Host response to open application info resource session – application info unavailable.	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>
29	Host response to open application info resource session – incorrect version of application info.	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>
30	Host response to open application info resource session – application info busy	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>
31	Host response to open application info resource session – invalid status byte	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card continues operation and will not open a session to the application info resource.	Minimum – Host reports error to user. Preferred – Applications on the Card may not operate correctly, including MMI. <sup>1</sup>



	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
32	Card requests to open conditional access session to the Host times out after 5 seconds.	Host	None	Minimum – Card sets IIR flag and stops responding to polls.	Host reports error to user. <sup>1</sup>
33	Card response to conditional access resource session – conditional access non-existent	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card will not descramble but will continue other operation and reports this in its MMI diagnostic application.	Minimum – Host reports error to user. Preferred – Scrambled channels are not viewed. <sup>1</sup>
34	Card response to conditional access resource session – conditional access unavailable	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card will not descramble but will continue other operation and reports this in its MMI diagnostic application.	Minimum – Host reports error to user. Preferred – Scrambled channels are not viewed. <sup>1</sup>
35	Card response to conditional access resource session – incorrect version of conditional access	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card will not descramble but will continue other operation and reports this in its MMI diagnostic application.	Minimum – Host reports error to user. Preferred – Scrambled channels are not viewed. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
36	Card response to conditional access resource session – conditional access busy	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card will not descramble but will continue other operation and reports this in its MMI diagnostic application.	Minimum – Host reports error to user. Preferred – Scrambled channels are not viewed. <sup>1</sup>
37	Card response to conditional access resource session – invalid status byte	Host	None	Minimum – Card sets IIR flag and stops responding to polls. Preferred – Card will not descramble but will continue other operation and reports this in its MMI diagnostic application.	Minimum – Host reports error to user. Preferred – Scrambled channels are not viewed. <sup>1</sup>
38	Card fails to respond to ca_info_inq within 5 seconds.	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful. Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	Host reports error to user.

	<b>Error Condition</b>	<b>Failure Mechanism</b>	<b>Host Action</b>	<b>SCTE Card Action</b>	<b>Comments</b>
39	Card requests to open copy protection resource session to the Host times out after 5 seconds.	Host	None	Minimum – Card continues operation, disables descrambling of all conditional access channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
40	Host response to open copy protection resource session – copy protection non-existent	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
41	Host response to open copy protection resource session – copy protection unavailable	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
42	Host response to open copy protection resource session – copy protection busy	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
43	Host response to open copy protection resource session – invalid status byte	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
44	Host does not support CableCARD device's copy protection system.	Host/Card incompatibility	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
45	Host and Card do not mate	Host/Card incompatibility	None	Minimum – Card continues operation, it SHALL enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
46	Host response to CP_sync – Host busy	Host	None	Minimum – Card will cease descrambling of copy protected channels.	A copy protected channel will stop being descrambled.

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
47	Host response to CP_sync – no CP support	Host	None	Minimum – Card will cease descrambling of copy protected channels.	A copy protected channel will stop being descrambled.
48	Host response to CP_sync – invalid status	Host	None	Minimum – Card will cease descrambling of copy protected channels.	A copy protected channel will stop being descrambled.
49	Host fails to respond to cp_open_req.	Host	None	Minimum – Card will cease descrambling of copy protected channels.	A copy protected channel will stop being descrambled.
50	Invalid Host certificate	Host	None	Minimum – Card continues operation, it SHALL NOT enable descrambling of any conditional access encrypted channels, reports to headend if possible, reports this to user, and reports this in its MMI diagnostic application.	All CA channels will not be descrambled, only clear channels may be viewed. <sup>1</sup>
51	Write Error (WE) occurs after completion of any transfer from Host to POD	Card or Host	Host performs Card reset.	None	User may see frozen picture on scrambled channels. <sup>1</sup>
52	Read Error (RE) occurs after completion of any transfer from Card to Host	Card or Host	Host performs Card reset.	None	User may see frozen picture on scrambled channels. <sup>1</sup>

	<b>Error Condition</b>	<b>Failure Mechanism</b>	<b>Host Action</b>	<b>SCTE Card Action</b>	<b>Comments</b>
53	Card fails to respond to any request within 5 seconds	Card	Minimum – Perform 1 PCMCIA reset, Report Error if not successful. Optional – Retry PCMCIA resets up to two times and report error. Preferred – Perform at least 1 PCMCIA reset. Report Error if not successful and continue to perform PCMCIA resets.	None	User MAY see frozen picture on scrambled channels.
54	Invalid session APDU from Host	Host	None	No action	Not a failure condition
55	Invalid session APDU from Card	Card	Host ignores invalid sessions.	None	Not a failure condition
56	Invalid SPDU tag from Host	Host	None	No action	Not a failure condition
57	Invalid SPDU tag from Card	Card	Host ignores invalid SPDU tags.	None	Not a failure condition
58	Invalid APDU tag from Host	Host	None	No action	Not a failure condition
59	Invalid APDU tag from Card	Card	Host ignores invalid APDU tags.	None	Not a failure condition
60	Transport ID from Host that has not been created and confirmed by Card	Host	None	No action	Not a failure condition
61	Transport ID from Card that has not been created by Host.	Card	Host ignores transport ID's that have not been created	None	Not a failure condition
62	Session ID from Host that has not been created and confirmed by Card	Host	None	No action	Not a failure condition
63	Session ID from CableCARD device that has not been created by Host.	Card	Host ignores session ID's that have not been created	None	Not a failure condition
64	Incompatible CableCARD device Inserted	Host	Reports error using screen in Figure B-1 - Error Display	None	Used when an S-CARD is inserted into an M-Host.

	Error Condition	Failure Mechanism	Host Action	SCTE Card Action	Comments
65	Card Resource Limit Reached	Card	Reports error using screen in Figure B-1 - Error Display	None	Used when the stream, program and/or PID limit has been reached by a user initiated action.

NOTE: A Card reset is defined as the Host's setting the RS bit in the command interface control register. A PCMCIA reset is defined as the Host's setting the RESET signal active on the PCMCIA interface.

1 - If the error is caused by an issue with the design of the Host or Card, this should be detected during certification.

In the event that an error occurs in which the Host must display an error message, the following message, or its equivalent, SHALL be displayed:

**A technical problem is preventing you  
from receiving all cable services at  
this time.**

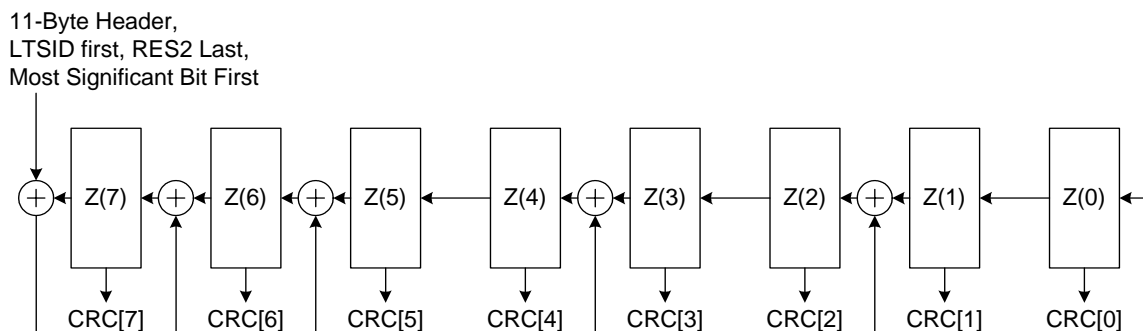
**Please call your cable operator and  
report error code 161-xx to have this  
problem resolved.**

**Figure B-1 - Error Display**

The “xx” after the error code 161 SHALL be the item number of the above table which has failed.

## Annex C CRC-8 Reference Model

The 8-bit CRC generator/checker for the Card Operating in M-Mode. is specified in Figure C-1.



**Figure C-1 - 8 bit CRC generator/checker model**

The model shown above implements the CRC-8 value used in the MPEG Transport Stream Pre-Header, utilizing the generator Polynomial:

$$x^8 + x^7 + x^6 + x^4 + x^2 + 1$$

The CRC-8 generator/checker operates on the first 11 bytes of the MPEG Transport Stream Pre-Header, starting with the LTSID field and ending with the RES2 field. Each byte is operated on Most Significant Bit first, and the model is initialized with all ones before the first byte is sent through the model. After the 11 bytes are processed, the CRC-8 value (CRC[7:0]) is taken from the 8 delay elements of the model. This value is placed in the 12<sup>th</sup> byte of the MPEG Transport Stream Pre-Header (for the generator) or compared with the 12<sup>th</sup> byte of the MPEG Transport Stream Pre-Header (for the checker).

An example stream and associated CRC-8 is:

0x01, 0x00, 0x55, 0xAA, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00

Produces a CRC of 0x8A.



## Annex D S-CARD Attribute and Configuration Registers

### D.1 General

The following sections are a detailed map of the attribute registers and configuration option register of the Card, also known as, SCTE Point of Deployment (POD) module. It is assumed that the reader is familiar with the PC Card tuple arrangement for the attribute registers.

### D.2 Attribute Tuples

The following is a list of the attribute tuples which SHALL be implemented in the Card/POD module.

CISTPL\_LINKTARGET  
 CISTPL\_DEVICE\_OA  
 CISTPL\_DEVICE\_OC  
 CISTPL\_VERS\_1  
 CISTPL\_MANFID  
 CISTPL\_CONFIG  
 CCST\_CIF  
 CISTPL\_CFTABLE\_ENTRY  
 STCE\_EV  
 STCE\_PD  
 CISTPL\_NO\_LINK  
 CISTPL\_END

#### D.2.1 CISTPL\_LINKTARGET

Defined in section 3.1.4 of [PCMCIA4], this is recommended by the PC Card standard for low voltage PC Cards for robustness. This would be in addition to the tuples defined in EIA 679-B Part B and would be the first tuple.

**Table D.2–1 - CISTPL\_LINKTARGET**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_LINKTARGET (0x13)							
1	02	TPL_LINK = 0x03							
2	04	TPL_TAG (3 bytes) = 0x43 (C)							
3	06	0x49 (I)							
4	08	0x53 (S)							

#### D.2.2 CISTPL\_DEVICE\_OA

Defined in section 3.2.3 of [PCMCIA4], this tuple is used to define the attribute memory operation.

**Table D.2–2 - CISTPL\_DEVICE\_0A**

Byte	Address (hex)	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_DEVICE_0A (0x1D )							
1	02	TPL_LINK = 0x04							
2	04	Other_Conditions_Info = 0x02							
3	06	Device_ID_1 = 0x08							
4	08	Device_Size = 0x00							
5	0A	0xFF							

**D.2.3 CISTPL\_DEVICE\_0C**

Defined in section 3.2.3 of [PCMCIA4], this tuple is used to define the common memory operation.

**Table D.2–3 - CISTPL\_DEVICE\_0C**

Byte	Address(hex)	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_DEVICE_0C (0x1C )							
1	02	TPL_LINK = 0x04							
2	04	Other_Conditions_Info = 0x02							
3	06	Device_ID_1 = 0x08							
4	08	Device_Size = 0x00							
5	0A	TPL_END = 0xFF							

**D.2.4 CISTPL\_VERS\_1**

Defined in section 3.2.10 of [PCMCIA4] with the exception that TPLLV1\_MAJOR be 0x05 and that TPLLV1\_MINOR = 0x00. The field name of the product SHALL be “OPENCABLE POD Module”.

**Table D.2–4 - CISTPL\_VERS\_1**

Byte	Address (hex)	7	6	5	4	3	2	1	0
0	00	TPL CODE = CISTPL_VERS_1 (0x15)							
1	02	TPL LINK = 26+n+m							
2	04	TPLL V1 MAJOR = 0x05							
3	06	TPLL V1 MINOR = 0x00							
4	08	TPPL V1 INFO = {Name of manufacturer (n bytes)}							
4+n	08+(2*n)	TPLL V1 INFO (multiple bytes) 0x00 (Null)							
5+n	0A+(2*n)	0x4F (O)							
6+n	0C+(2*n)	0x50 (P)							
7+n	0E+(2*n)	0x45 (E)							
8+n	10+(2*n)	0x4E (N)							
9+n	12+(2*n)	0x43 (C)							
10+n	14+(2*n)	0x41 (A)							
11+n	16+(2*n)	0x42 (B)							
12+n	18+(2*n)	0x4C (L)							
13+n	1A+(2*n)	0x45 (E)							
14+n	1C+(2*n)	0x20 ( )							
15+n	1E+(2*n)	0x50 (P)							
16+n	20+(2*n)	0x4F (O)							
17+n	22+(2*n)	0x44 (D)							
18+n	24+(2*n)	0x20 ( )							
19+n	26+(2*n)	0x4D (M)							
20+n	28+(2*n)	0x6F (o)							
21+n	2A+(2*n)	0x64 (d)							
22+n	2C+(2*n)	0x75 (u)							
23+n	2E+(2*n)	0x6C (l)							
24+n	30+(2*n)	0x65 (e)							
25+n	32+(2*n)	0x00 (Null)							
26+n	34+(2*n)	Additional Product Information (m bytes)							
27+n	36+(2*n)	0x00 (Null)}							
27+n+m	36+(2*n)+m	TPL_END = 0xFF							

### D.2.5 CISTPL\_MANFID

Defined in section 3.2.9 of [PCMCIA4].

**Table D.2-5 - CISTPL\_MANFID**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_MANFID (0x20)							
1	02	TPL_LINK = Link to next tuple (at least 4)							
2	04	TPLMID_MANF = PC Card manufacturer code							
3	06	TPLMID_CARD = manufacturer information (Part Number and/or Revision)							

### D.2.6 CISTPL\_CONFIG

Defined in section 3.3.4 of [PCMCIA4].

**Table D.2-6 - CISTPL\_CONFIG**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_CONFIG (0x1A)							
1	02	TPL_LINK = 5+n+m+p							
2	04	0		TPCC_RMSZ				TPCC_RASZ	
3	06	0		TPCC_LAST					
4	08	n bytes of TPCC_RADR							
5+n	0A+(2*n)	m bytes of TPCC_RMSK							
6+n+m	0C+(2* (n+m))	19 bytes of TPCC_SBTPL							
25+n+m	32+(2* (n+m+p))	TPL_END = 0xFF							

TPCC_RMSZ	The number of bytes in the configuration registers Base Address in Attribute Memory Space field (TPCC_RMSK) of this tuple is the value of this field plus 1. For the Card, this value will depend on the manufacturer.
TPCC_RASZ	The number of bytes in the Configuration Register presence mask field (TPCC_RADR field) of the tuple is this value plus 1. For the Card, this value will depend on the manufacturer.
TPCC_LAST	One byte field which contains the Configuration Index Number of the last configuration described in the Card Configuration Table. Once the Host encounters this configuration, when scanning for valid configurations, it SHALL have processed all valid configurations. For the Card, this value will depend on the manufacturer.
TPCC_RADR	The Base Address of the Configuration Registers, in an even byte of Attribute Memory (address of Configuration Register 0), is given in this field. This Address SHALL NOT be greater than 0xFFE.
TPCC_RMSK	The presence mask for the Configuration Registers is given in this field. Each bit represents the presence (1) or absence (0) of the corresponding Configuration Register.
TPCC_SBTPL	The sub-tuple allows for additional configuration sub-tuples. The CCST_CIF sub-tuple SHALL be implemented.

### D.2.7 CCST-CIF

Defined in section 3.3.4.5.1 of [PCMCIA4]. The interface ID number (STCI\_IFN) is 0x41. STCI\_STR is defined to be 'OpenCable\_POD\_V1.00'.

**Table D.2-7 - CCST-CIF**

Byte	Address <sub>H</sub>	7	6	5	4	3	2	1	0
0	00	ST_CODE = CCST_CIF (0xC0)							
1	02	ST_LINK = 0x0B							
2	04	STCI_IFN = 0x41							
3	06	STCI_IFN_1 = 0x03							
4	08	STCI_STR (multiple bytes) 0x50 (P)							
5	0A	0x4F (O)							
6	0C	0x44 (D)							
7	0E	0x5F ( )							
8	10	0x56 (V)							
9	12	0x31 (1)							
10	14	0x2E (.)							
11	16	0x30 (0)							
12	18	0x30 (0)							
13	1A	0x00 (Null)							
14	1C	TPL_END 0xFF							

### D.2.8 CISTABLE\_ENTRY

Defined in section 3.3.2 of [PCMCIA4]. For the first entry TPCE\_IND<sub>X</sub> has both bits 6 (Default) and 7 (Interface) set. The Configuration Entry Number is selected by the manufacturer. TPCE\_IF = 0x04 – indicating Custom Interface 0. TPCE\_FS SHALL indicate the presence of both I/O and power configuration entries. TPCE\_IO is a 1-byte field with the value 0x22. The information means: 2 address lines are decoded by the Card and it uses only 8-bit accesses. The power configuration entry – required by this specification, SHALL follow the PC Card Specification.” Additionally, two sub-tuples, STCE\_EV and STCE\_PD SHALL be included.

The power descriptor for V<sub>cc</sub> is modified to 1 A.

**Table D.2-8 - CISTPL\_CFTABLE\_ENTRY**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_CFTABLE_ENTRY (0x1B)							
1	02	TPL_LINK == 0x33							
2	04	TPCE_IND <sub>X</sub> = 0xC0 LOGICAL OR Config. Entry Number <sub>H</sub>							
3	06	TPCE_IF = 0x04							
4	08	TPCE_FS = 0x0A							
5	0A	TPCE_PD V <sub>cc</sub> Parameter Selection Byte = 0x38							
6	0C	TPCE_PD V <sub>cc</sub> Static Current = Manufacturer value							
7	0E	TPCE_PD V <sub>cc</sub> Average Current = 0x07							
8	10	TPCE_PD V <sub>cc</sub> Peak Current = 0x07							
9	12	TPCE_PD V <sub>pp</sub> Parameter Selection Byte = 0x78							
10	14	TPCE_PD V <sub>pp</sub> Static Current = Manufacturer value							
11	16	TPCE_PD V <sub>pp</sub> Average Current = 0x26							
12	18	TPCE_PD V <sub>pp</sub> Peak Current = 0x26							
13	1A	TPCE_PD V <sub>pp</sub> Power Down Current = Manufacturer value							
14	1C	TPCE_IO = 0x22							
15	1E	ST_CODE = STCE_EV (0xC0)							
16	20	ST_LINK = 0x10							
17	22	STEV_STRS = “NRSS_HOST” 0x4F (O)							
18	24	0x50 (P)							
19	26	0x45 (E)							
20	28	0x4E (N)							

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
21	2A	0x43 (C)							
22	2C	0x41 (A)							
23	2E	0x42 (B)							
24	30	0x4C (L)							
25	32	0x45 (E)							
26	34	0x5F ( )							
27	36	0x48 (H)							
28	38	0x4F (O)							
29	3A	0x53 (S)							
30	3C	0x54 (T)							
31	3E	0x00 (Null)							
32	40	0xFF							
33	42	ST_CODE = STCE_PD (0xC1)							
34	44	ST_LINK = 0x12							
35	46	STPD_STRS = “NRSS CI MODULE” 0x45 (O)							
36	48	0x50 (P)							
37	4A	0x45 (E)							
38	4C	0x4E (N)							
39	4E	0x43 (C)							
40	50	0x41 (A)							
41	52	0x42 (B)							
42	54	0x4C (L)							
43	56	0x45 (E)							
44	58	0x5F( )							
45	5A	0x4D (M)							
46	5C	0x4F (O)							
47	5E	0x44 (D)							
48	60	0x55 (U)							
49	62	0x4C (L)							
50	64	0x45 (E)							
51	66	0x00 (Null)							
52	68	0xFF							
53	6A	0xFF							

### D.2.9 STCE\_EV

Defined in section 3.3.2.10.1 of [PCMCIA4]. Only the system name is ‘OPENCABLE\_HOST’.

**Table D.2–9 - STCE\_EV**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	ST_CODE = STCE_EV (0xC0)							
1	02	ST_LINK = Link to next tuple (at least m-1)							
2	04	STPD_STRS = A list of strings, the first being ISO 646 coded, and the rest being coded as ISO alternate language strings, with the initial escape character suppressed. Each string is terminated by a 0 byte, and the last string, if it does not extend to the end of the subtuple, is followed by a 0xff byte.							

### D.2.10 STCE\_PD

Defined in section 3.3.2.10.2 of [PCMCIA4]. Only the physical device name is ‘OPENCABLE\_POD\_MODULE’.

**Table D.2–10 - STCE\_PD**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	ST_CODE = STCE_PD (0xC1)							
1	02	ST_LINK = Link to next tuple (at least m-1)							
2	04	STPD_STRS = A list of strings, the first being ISO 646 coded, and the rest being coded as ISO alternate language strings, with the initial escape character suppressed. Each string is terminated by a 0 byte, and the last string, if it does not extend to the end of the subtuple, is followed by a 0xff byte.							

### D.2.11 CISTPL\_END

Defined in section 3.1.2 of [PCMCIA4]. If the CA Card contains other tuples in addition to those defined above then these will come before CISTPL\_END.

**Table D.2–11 - CISTPL\_END**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_END(0xFF)							

## D.3 Configuration Option Register

Defined in section 4.15.1 of [PCMCIA2].

**Table D.3–1 - Configuration Option Register**

Byte	Address <sub>(hex)</sub>	7	6	5	4	3	2	1	0
0	00	SRESET	LevIRE Q	Function Configuration Index					

## D.4 Values to Enable CableCARD Personality Change

SRESET – 0 (Do not soft reset (POD reset) the Card)

LevIREQ – 1 (Card generates Level Mode interrupts).

Function Configuration Index – Lower 6 bits of TPCE\_IND.

## D.5 Operation After Invoking CableCARD Personality Change

After the correct value is written into the configuration register, the Card SHALL wait a minimum of 10 usec before switching from the PCMCIA to the Card interface.

## **Annex E   DownloadInfoIndicator Message Detail for Common Download (Normative)**

The information in this section is given here to facilitate Common Download. This information is to clarify ambiguities in the DSM-CC specification for use for Common Download. Since the code objects are carried in a DSM-CC data carousel, a standard format for the DownloadInfoIndicator (DII) message should be defined so that Host device can interpret this message properly.

The DownloadInfoIndication (DII) message will be defined as identified in Table E-1.



**Table E-1 - DownloadInfoIndicator Message Detail**

Syntax	No. of Bits	Mnemonic
DSMCC_section() {		
Table_id	8	uimbsbf
section_syntax_indicator	1	bslbf
reserved	3	bslbf
dsmcc_section_length	12	uimbsbf
table_id_extension	16	uimbsbf
reserved	2	bslbf
version_number	5	uimbsbf
current_next_indicator	1	bslbf
section_number	8	uimbsbf
last_section_number	8	uimbsbf
dsmccMessageHeader() {		
protocolDiscriminator	8	uimbsbf
DsmccType	8	uimbsbf
MessageId	16	uimbsbf
TransactionId () {		
Update_Flag	1	bslbf
transaction_identifier	15	uimbsbf
Version_number	14	uimbsbf
Originator	2	uimbsbf
}		
Reserved	8	
adaptionLength	8	uimbsbf
messageLength	16	uimbsbf
}		
DownloadInfoIndication() {		
DownloadId	32	uimbsbf
blockSize	16	uimbsbf
windowSize	8	uimbsbf
ackPeriod	8	uimbsbf
tCDownloadWindow	32	uimbsbf
tCDownloadScenario	32	uimbsbf
compatibilityDescriptor()	16	uimbsbf
numberOfModules	16	uimbsbf
for (I=0; i<numberOfModules; i++){		
ModuleID	16	uimbsbf
ModuleSize	32	uimbsbf
ModuleVersion	8	uimbsbf
ModuleInfoLength	8	uimbsbf
for (i=0; i<ModuleInfoLength; i++){		
ModuleInfoByte	8	uimbsbf
}		
}		
privateDataLength	16	uimbsbf
for (i=0; i<privateDataLength; i++){		
privateDataByte	8	uimbsbf
}		
}		
CRC_32	32	uimbsbf

**Table\_ID** Expected value SHALL be 0x3B for the DII table

**section\_syntax\_indicator** Expected Value SHALL be 0x01

<b>dsmcc_section_length</b>	The number of bytes in the messages following this byte
<b>table_id_extension</b>	The table_id_extension field is populated from the bottom two bytes of the transaction_id field from the dsmccMessageHeader section of the message.
<b>version_number</b>	The version_id field is populated from bits 16..20 of the transaction_id field from the dsmccMessageHeader section of the message. Generally both of these fields should be 0 per the ISO/IEC 13818-6 Specification.
<b>current_next_indicator</b>	This one bit field matches the MSB of the transaction_id field from the dsmccMessageHeader section of the message.
<b>section_number</b>	Expected value SHALL be 0x00 for this message
<b>last_section_number</b>	The expected value SHALL be 0x00 for this message
<b>protocolDiscriminator</b>	The expected value SHALL be 0x0b
<b>DsmccType</b>	The expected value SHALL be 0x03
<b>MessageId</b>	The expected value SHALL be 0x1002, to denote a DII message
<b>Update_Flag</b>	Used to determine if the message has been updated or not
<b>transaction_identifier</b>	Used for session integrity and error processing. It should be a unique number.
<b>Version_number</b>	The expected value SHALL be 0x00
<b>Originator</b>	The expected value SHALL be 0x02. To denote the DSM-CC User Id adaption format.
<b>adaptionLength</b>	The expected value SHALL be 0x00
<b>messageLength</b>	Length of the DownloadInfoIndication part of the message.
<b>Download_ID</b>	The download_id for this message SHALL be 0x03E8 or 1000 decimal.
<b>blockSize</b>	Is dependent on the size of the module blocks being sent out. This needs to match the module blocks size.
<b>windowSize</b>	The expected value SHALL be 0x00
<b>ackPeriod</b>	The expected value SHALL be 0x00
<b>tCDownloadWindow</b>	The expected value SHALL be 0x00
<b>tCDownloadScenario</b>	Indicates the timeout period in microseconds for the entire download scenario in progress.
<b>compatibilityDescriptor</b>	This structure SHALL only contain the compatibilityDescriptorLength field of the compatibilityDescriptor as defined in DSM-CC. It SHALL be set to the value 0x0000.
<b>numberOfModules</b>	The number of modules described in the loop following this field. The loop includes information for all of the modules to be downloaded by the Client. For the Data Carousel scenario, the loop describes a subset of all the modules associated with this Data Carousel, although it may describe all of them.
<b>moduleID</b>	An identifier for the module that is described by the moduleSize, moduleVersion and moduleInfoByte field. The moduleID is unique within the scope of the downloadID.
<b>moduleSize</b>	The length in bytes of the described module.
<b>moduleVersion</b>	The version of the described module.
<b>moduleInfoLength</b>	The length in bytes of the moduleInfo field.

<b>moduleInfoByte</b>	Conveys a list of descriptors. Each list will define one or more attributes of the associated modules. This field contains the ASCII representation of the code_file_name.
<b>privateDataLength</b>	Not used within the scope of these messages at this time.
<b>privateDataByte</b>	Not used within the scope of these messages at this time.

## Appendix I Revision History

The following ECNs were incorporated into OC-SP-CCIF2.0-I02-050708:

ECN	Description	Date
CCIF2.0-N-05.0769-6	Modifications to extended channel resource to account for eCM	7/1/05
CCIF2.0-N-05.0782-1	Multi-stream Homing Modification	6/17/05
CCIF2.0-N-05.0787-2	Omnibus ECR	6/17/05
CCIF2.0-N-05.0788-3	DownloadInfoIndicator Message Detail for Common Download	6/17/05