OpenCable™ Specifications

CableCARD Interface 2.0 Specification

OC-SP-CCIF2.0-I26-130418

ISSUED

DISCLAIMER

This document is published by Cable Television Laboratories, Inc. ("CableLabs®").

CableLabs reserves the right to revise this document for any reason including, but not limited to, changes in laws, regulations, or standards promulgated by various agencies; technological advances; or changes in equipment design, manufacturing techniques, or operating procedures described, or referred to, herein. CableLabs makes no representation or warranty, express or implied, with respect to the completeness, accuracy, or utility of the document or any information or opinion contained in the report. Any use or reliance on the information or opinion is at the risk of the user, and CableLabs shall not be liable for any damage or injury incurred by any person arising out of the completeness, accuracy, or utility of any information or opinion contained in the document.

This document is not to be construed to suggest that any affiliated company modify or change any of its products or procedures, nor does this document represent a commitment by CableLabs or any cable member to purchase any product whether or not it meets the described characteristics. Nothing contained herein shall be construed to confer any license or right to any intellectual property, whether or not the use of any information herein necessarily utilizes such intellectual property. This document is not to be construed as an endorsement of any product or company or as the adoption or promulgation of any guidelines, standards, or recommendations.

Document Status Sheet

Document Control Number:	OC-SP-CCIF2.0-I26-130418			
Document Title:	CableCARD Interface 2.0 Specification			
Revision History:	I01 - March 31, 2005 I02 - July 8, 2005 I03 - November 17, 2005 I04 - January 26, 2006 I05 - April 13, 2006 I06 - June 22, 2006 I07 - August 3, 2006 I08 - October 31, 2006 I09 - January 5, 2007 I10 - March 23, 2007 I11 - June 15, 2007 I12 - November 13, 2007 I13 - January 18, 2008		I14 - April 4, 2008 I15 - June 20, 2008 I16 - November 14, 2008 I17 - February 6, 2009 I18 - May 8, 2009 I19 - September 4, 2009 I20 - December 11, 2009 I21 - May 7, 2010 I22 - September 10, 2010 I23 - May 12, 2011 I24 - January 12, 2012 I25 - May 31, 2012 I26 - April 18, 2013	
Date:	April 18, 2013			
Status:	~~Work in Progress~~	~~Draft~~	Issued	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/Member/ Vendor~~	Public

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	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.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Contents

Tables

Figures

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 [SCTE28], the Host-POD Interface Standard, and [SCTE41], 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 [SCTE28] documents. For the text that was extracted from the [SCTE28] 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.

- 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).
- 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 [SCTE28], 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 [SCTE28].

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"	This word means that the item is an absolute requirement of this specification.
"SHALL NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word 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 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.

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.

All references are subject to revision, and parties to agreement based on this specification are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific:

- For a specific reference, subsequent revisions do not apply.
- For a non-specific, non-Bundle reference, the latest version applies.
- For non-specific CableLabs references that are part of the [OC-BUNDLE], the versions mandated in a particular Bundle apply.

2.1 Normative References

[CCCP]	CableCARD Copy Protection 2.0 Specification, OC-SP-CCCP2.0, Cable Television Laboratories, Inc. Referenced in [OC-BUNDLE].
[CDL]	Common Download 2.0, OC-SP-CDL2.0, Cable Television Laboratories, Inc. Referenced in [OC-BUNDLE].
[DOCSIS2.0]	Data-Over-Cable Service Interface Specifications, DOCSIS 2.0, Radio Frequency Interface Specification, CM-SP-RFIv2.0-C02-090422, April 22, 2009, Cable Television Laboratories, Inc.
[DSG]	DOCSIS Set-top Gateway (DSG) Interface Specification, CM-SP-DSG-I23-130404, April 4, 2013, Cable Television Laboratories, Inc.
[HOST]	OpenCable Host Device 2.1 Core Functional Requirements, OC-SP-HOST2.1, Cable Television Laboratories, Inc. Referenced in [OC-BUNDLE].
[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: 1998 Codes for the representation of names of Languages - Part 2: 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.
[J042]	American National Standard, J-STD-042-A-2007, Emergency Alert Message for Cable (SCTE 18 and EIA/CEA 814).

[OC-BUNDLE]	OpenCable Bundle Requirements, OC-SP-BUNDLE. See Section 2.3.1 to acquire this specification.
[OC-SEC]	OpenCable System Security Specification, OC-SP-SEC, Cable Television Laboratories, Inc. Referenced in [OC-BUNDLE].
[OCAP]	OpenCable Application Platform (OCAP), OC-SP-OCAP, Cable Television Laboratories, Inc. Referenced in [OC-BUNDLE].
[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.
[RFC1901]	IETF RFC 1901, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Introduction to Community-based SNMPv2", January 1996.
[RFC1902]	IETF RFC 1902, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", January 1996.
[RFC2131]	IETF RFC 2131, R. Droms, "Dynamic Host Configuration Protocol", March 1997.
[RFC2132]	IETF RFC 2132, DHCP Options and BOOTP Vendor Extensions, March 1997.
[RFC2396]	IETF RFC 2396, T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", August 1998.
[RFC3415]	IETF RFC 3415, B. Wijnen, R. Presuhn, K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP"), December 2002.
[SCTE23-2]	ANSI/SCTE 23-2 2012, DOCSIS 1.1 Part 2 Baseline Privacy Interface Plus.
[SCTE28]	ANSI/SCTE 28 2007, 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]	CEA-679-C Part B, National Renewable Security Standard (July 2005). A joint work of NCTA and CEMA Technology and Standards.

2.3 Reference Acquisition

2.3.1 OpenCable Bundle Requirements

The OpenCable Bundle Requirements specification [OC-BUNDLE] indicates the set of CableLabs specifications required for the implementation of the OpenCable Bundle. The version number of [OC-BUNDLE] corresponds to the release number of the OpenCable Bundle that it describes. One or more versions of [OC-BUNDLE] reference

this specification. Current and past versions of [OC-BUNDLE] may be obtained from CableLabs at http://www.cablelabs.com/opencable/specifications.

2.3.2 Other References

- ***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 Varembé, Case postale 56, CH-1211 Geneva 20, Switzerland; Internet: http://www.iso.ch/

- ***SCTE Specifications***

  SCTE - Society of Cable Telecommunications Engineers Inc., 140 Philips Road, Exton, PA 19341;
  Telephone: +1-610-363-6888 / +1-800-542-5040; Fax: +1-610-363-5898; Internet: http://www.scte.org/

- ***ANSI/EIA Standards***

  American National Standards Institute, Customer Service, 11 West 42nd Street, New York, NY 10036;
  Telephone +1-212-642-4900; Facsimile +1-212-302-1286; E-mail: 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: +1-800-854-7179; Facsimile: +1-303-397-2740; E-mail: global@ihs.com; URL: <http://global.ihs.com>

- ***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: +1-703-620-8990; Facsimile: +1-703-620-9071; E-mail: ietf-secretariat@ietf.org; URL: http://www.ietf.org/rfc

3 TERMS AND DEFINITIONS

This specification uses the following terms:

American Standard Code for Information Interchange	Internationally recognized method for the binary representation of text.
Application Protocol Data Unit	A common structure to send application data between the Card and Host.
Application Program Interface	The software interface to system services or software libraries. An API can consist of classes, function calls, subroutine calls, descriptive tags, etc.
CableCARD™ device	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.
Card	CableCARD device.
Card Information Structure	Low-level configuration information contained in the Card's Attribute Memory.
Command Channel	Also identified as Data Channel.
Conditional Access and encryption	A system that provides selective access to programming to individual customers.
Conditional Access System	Secures delivery of cable services to the Card.
CPU Interface	The logical interface between the Card and the Host comprised of the Data and Extended communications channels.
Data Channel	Also identified as Command Channel.
DOCSIS Set-top Gateway	A method of using DOCSIS protocols to support a one-way out-of-band communication path.
Downstream	Transmission from headend to Host.
DSG Advanced Mode	Also known as Advanced DSG (ADSG). 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.
DSG SEB Client	DSG capable device that cannot establish a DOCSIS upstream connection, which therefore uses a DSG SEB Server to obtain DOCSIS interactive capabilities.
DSG SEB Server	DSG capable device that provides DOCSIS interactive capabilities to DSG SEB Clients residing on a shared home network interface, where the services include exposing of the DSG SEB Server's eCM such that the DSG SEB Client is able to acquire IP connectivity by way of the DSG SEB Server.
DSG Tunnel	A single instance of a DSG Rule within a DCD message.
Dynamic Host Configuration Protocol	An Internet standard for assigning IP addresses dynamically to IP hosts.
eCM	A DOCSIS Cable Modem that has been embedded into a Set-top/Host device and includes DSG functionality.
Encryption Mode Indicator	Defines the copy protection mode for digital outputs.
Entitlement Management Message	A conditional access control message to a Card.
Extended Application Information Table	Used for launching and managing the lifecycle of unbound applications for OCAP.

Extended Text Table	An MPEG 2 table contained in the Program and System Information Protocol ("PSIP"), which provides detailed descriptions of virtual channels and events.
Forward Application Transport	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.
Forward Data Channel	An out-of-band ("OOB") data channel from the headend to the Host.
Gapped Clock	A periodic signal in which some transitions are omitted creating gaps in a clock pattern.
Headend	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.
High-Z	Greater than 100K Ohm resistance to power or to ground.
Host	The consumer device used to access and navigate cable content. Typically a digital TV or set-top DTV receiver, further defined in [HOST].
HTTP	The transport layer for HTML documents over the Internet Protocol ("IP").
Hypertext Markup Language	A presentation language for the display of multiple media contents, typically used on the Internet.
Inband	Within the main Forward Applications Transport channel.
Internet Protocol	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.
IP datagram	An Internet Protocol datagram, which is either sent in a single MAC frame or may be fragmented and transmitted across multiple MAC frames.
IP packet	The portion of an Internet Protocol datagram inserted into or extracted from a MAC frame. If an IP datagram has been fragmented, then each of the fragments is an IP packet. If an IP datagram has not been fragmented, then the entire datagram is in a single IP packet.
IP Unicast	Point-to-Point Internet Protocol datagram service.
IP Multicast	Point to multi-point Internet Protocol datagram service.
Local Transport Stream ID	Assigned by the Host operating in M-Mode.
LSB	Least Significant Bit or Byte of a specified binary value.
M-CARD	Multi-Stream Card, capable of operating in either S-Mode or M-Mode.
M-Host	A Host device that implements the Multi-Stream Card/Host interface. This device may contain multiple FAT tuners but is not required to.
M-Mode	A Card or Host utilizing the Multi-Stream Card/Host interface is said to be operating in M-Mode.
Model ID	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).
MPEG	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.
MPEG-2 Video	ISO-IEC 13818-2, international standard for the compression of video.

MPEG-2 Transport	ISO-IEC 13818-1, international standard for the transport of compressed digital media.
MSB	Most Significant Byte or Bit, of a specified binary value.
National Television Systems Committee	An entity that developed the analog television system used in North America and elsewhere.
OpenCable Bundle	The OpenCable Bundle defines a set of specifications required to build a specific version of an OpenCable device. See [OC-BUNDLE].
OC Signaling	A DSG Broadcast Tunnel containing CVTs and XAITs as defined in [DSG].
Out-of-Band	The combination of the Forward and Reverse Data Channels. The OOB channel provides a data communication channel between the cable system and the Host.
PC Card	A device that complies with the PC Card Standard, as referenced in this document.
Point of Deployment	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.
Protocol Data Unit	A packet of data passed across a network or interface.
Quadrature Amplitude Modulation	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), 64-QAM (six bits), and 256-QAM (eight bits).
Quadrature Phase Shift Keying	A digital modulation method in which the state of a two-bit symbol is represented by one of four possible phase states.
Remote Procedure Call	The ability for client software to invoke a function or procedure call on a remote server machine.
Resource	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.
Return Data Channel	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.
S-CARD	Single-Stream Card compliant to ANSI/SCTE 28, capable of only operating in S-Mode.
Set-top Extender Bridge	A client/server architecture that allows DSG Set-tops, incapable of establishing a DOCSIS upstream connection thru the eCM, to forward IP traffic over an alternate connection utilizing a Home Network Interface.
S-Host	Single-Stream Host device.
S-Mode	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).
Subtuple	Subset of a Tuple.
Tuple	Data stored within a PC Card that can be used to determine the capabilities of the card.
Uniform Resource Locator	A standard method of specifying the location of an object or file.
Upstream	Transmission from host to headend.

User Datagram Protocol	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.
Virtual Channel Table	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:

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

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

PID	Packet Identifier
PIN	Personal Identification Number
PNG	Portable Network Graphics
POD	Point of Deployment
PPV	Pay-Per-View
PSI	Program Specific Information
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RDC	Return Data Channel
RF	Radio Frequency
RFC	Request For Comments
RPC	Remote Procedure Call
ROM	Read Only Memory
RX	Receive
SAS	Specific Application Support
SCTE	Society of Cable Telecommunications Engineers
SEB	Set-top Extender Bridge
SEBC	DOCSIS Set-top Gateway SEB Client
SEBS	DOCSIS Set-top Gateway SEB Server
SI	System Information
SRAM	Static Random Access Memory
tcimsbf	Two's complement integer, (msb) sign bit first
TPDU	Transport Protocol Data Unit
TSID	Transport Stream Identifier
TX	Transmit
UDP	User Datagram Protocol
uimsbf	Unsigned Integer Most Significant Bit First
URL	Uniform Resource Locator
V	Volt
VCR	Video Cassette Recorder
VCT	Virtual Channel Table
VOD	Video-on-Demand
WKMA	Well Known MAC Address
XAIT	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 either over the ***Extended Channel*** as described in Section 10 of this document or directly from the eCM via the DSG interface. 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 Generic IPPV resource. The CableCARD Interface support of the Generic IPPV resource has been deprecated. The preferred approach for supporting IPPV is to use the appropriate OCAP application. If the Host elects to use Generic IPPV resource, it should comply with Section 8.10 of [SCTE28].

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):
  - [SCTE55-1] Digital Broadband Delivery System: Out Of Band Transport Part 1: Mode A
  - [SCTE55-2] 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:

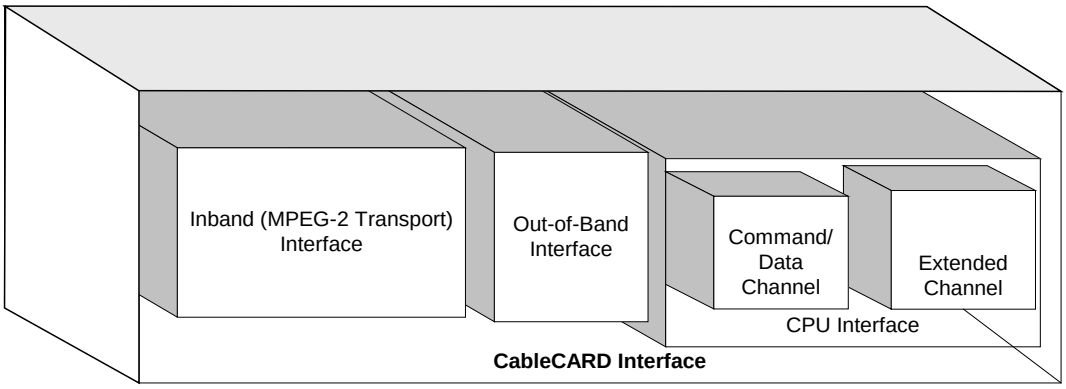


Figure 5.2-1 - Card Interfaces

Copy protection is required for protection of high-valued content, content marker with a non-zero EMI, across the CHI. Section 8, [CCCP], 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 7.3.5.1). The transport stream interface layers are shown below. The Transport Layer and all upper layers are defined in the MPEG-2 specification, ISO 13818.

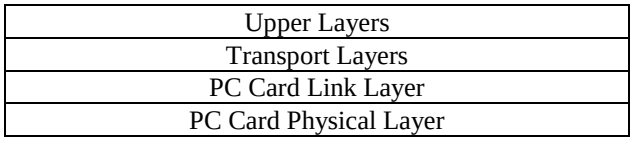


Figure 5.2-2 - Transport Stream Interface Layers

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.

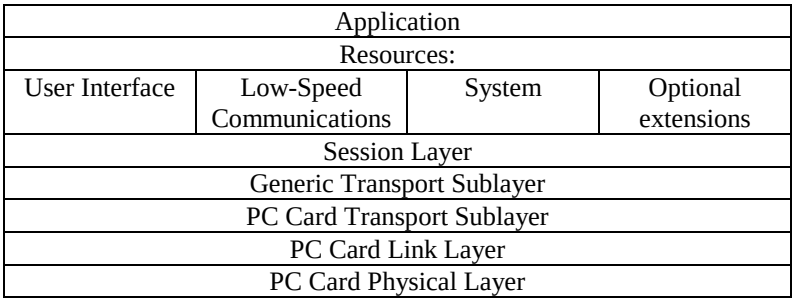


Figure 5.2-3 - Command Interface Layers

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 OOB signaling is utilized for CableCARD operational modes: 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 can 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
S-CARD	S-Mode	Host may reject S-CARD*
M-CARD	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 functions 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 functions in an M-Host built in compliance with M-Mode operation as defined in this document, capable of processing multiple transport streams.

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 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.

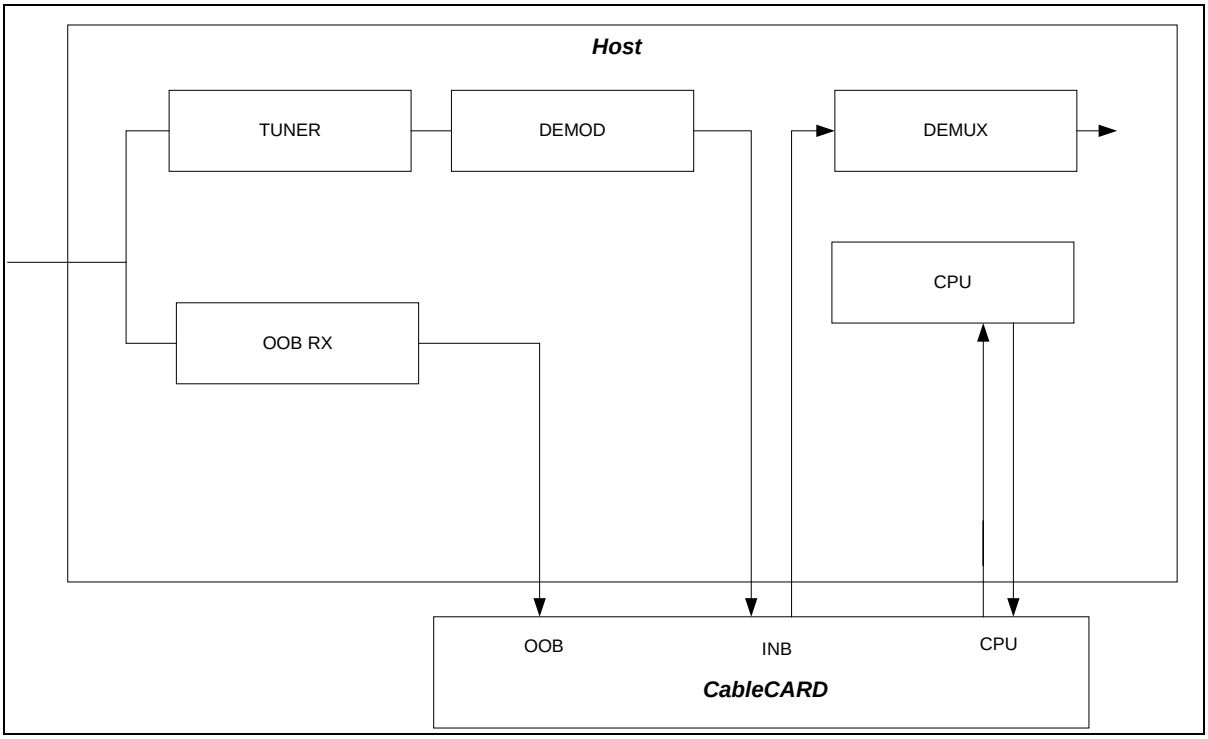


Figure 5.5-1 - System with One-way Network

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].

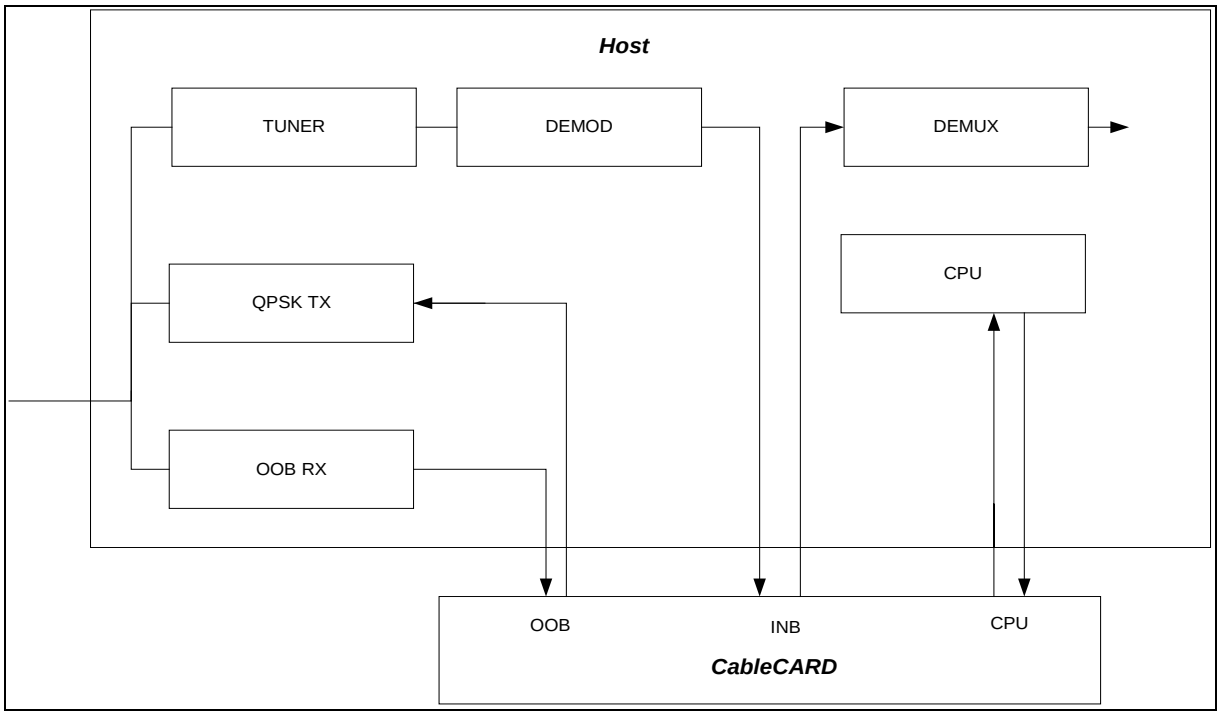


Figure 5.6-1 - System with Two-way Network

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 QPSK transmitter 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.

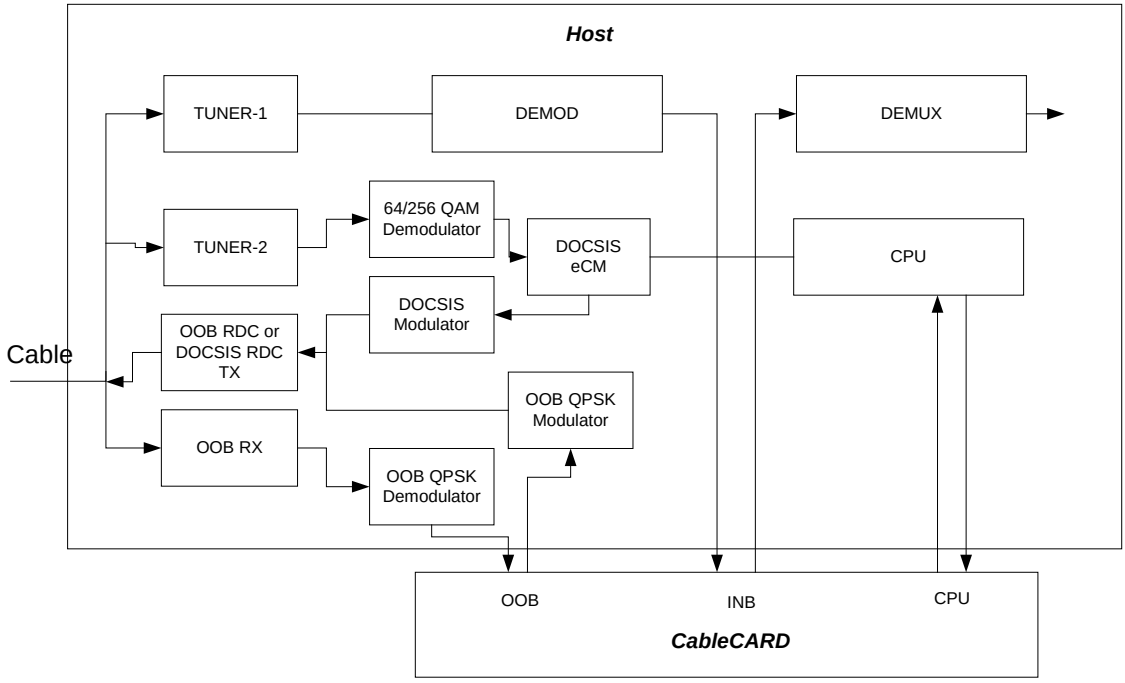


Figure 5.7-1 - System with DOCSIS Two-way Network

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 Two-way Networks with Set-top Extender Bridge (SEB)

The configuration shown in Figure 5.8-1 applies where DSG capability via extender bridge with another DSG capable device resides on the home network, as per the SEB architecture defined in [DSG].

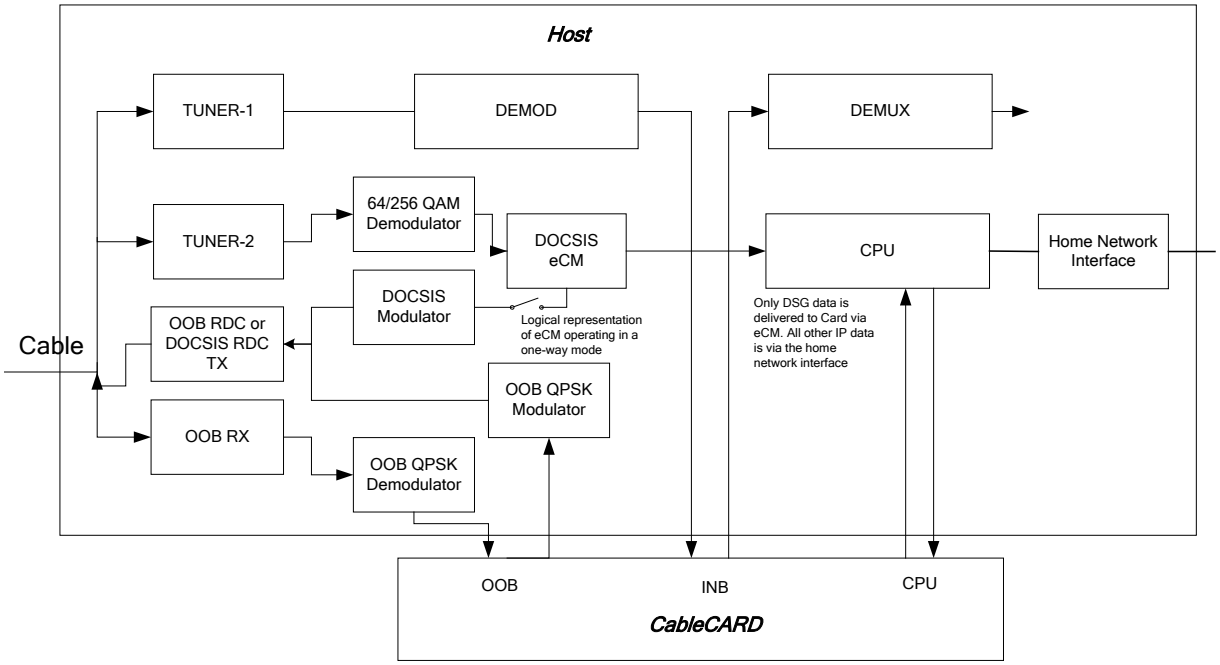


Figure 5.8-1 - System with SEB Two-way Network

In this configuration, a single upstream transmit path, where the actual upstream DOCSIS transmitter resides on SEB Server as per [DSG], is shared between the Card and the DOCSIS modem of the SEB Server. As in the previous configurations, the system will operate in one of two modes, where the DSG mode is such that the Host is utilizing SEB and operating as an SEB Client. Note that when operating as an SEB Client, the Card is completely unaware, as the APDUs and transactions performed by the Card are the exact same as if the Host were operating in a pure DSG mode.

- 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]. All other downstream IP messaging is delivered to the Host and subsequently passed to the Card via the home network interface, where the IP messages are being proxied by the SEB Server. The upstream Conditional Access Messages and network management messages will be transmitted from the Card via IP over the home network to the SEB Server, which in turn utilizes its DOCSIS upstream channel. The Card in this case utilizes the Extended Channel as it would when the Host is operating in a pure DSG mode. The DSG channel and the bi-directional channel provided by the home network interface can be used by any applications running in the Host, simultaneously with the Card's communication with the headend via the Extended Channel. 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 and whether or not the Host supports SEB and there is a SEB Server residing within the home. The Card informs the Host which of these modes is to be used, as detailed later in this specification.

5.9 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.

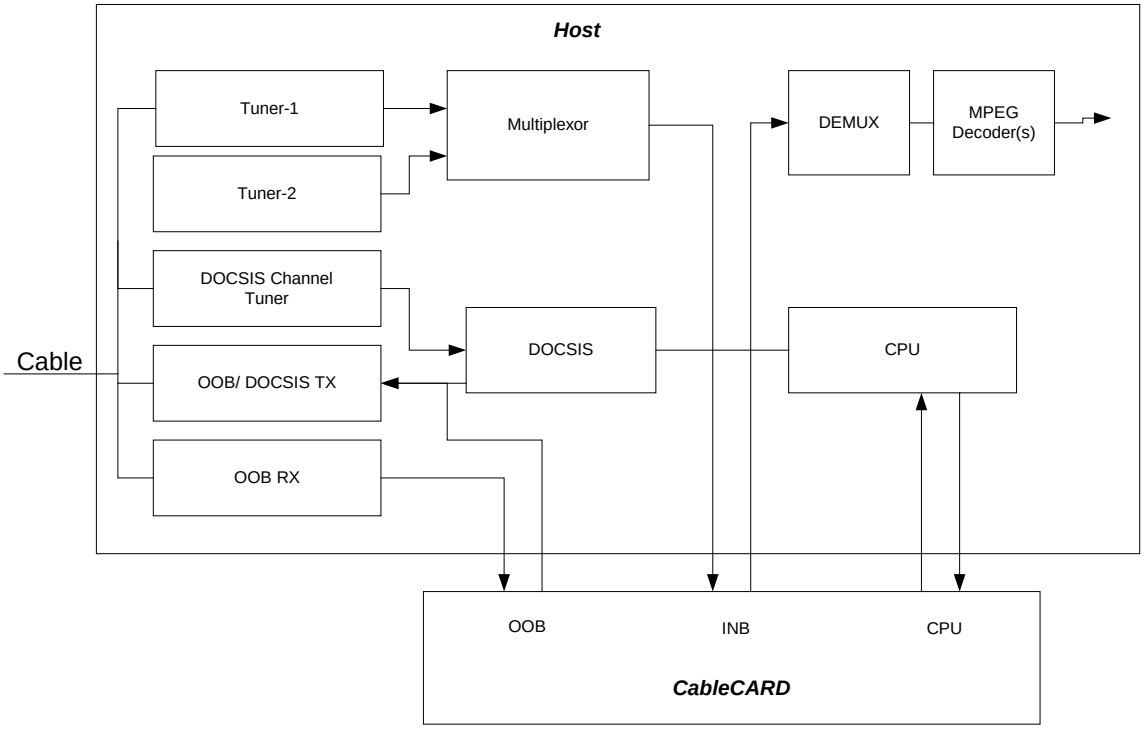


Figure 5.9-1 - Host and M-CARD Device Block Diagram Example

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 256-QAM tuners/demodulators, or up to six streams from six 64-QAM tuners/demodulators.

For the Card, the maximum number of unique MPEG transport streams input into the Card from the Host can 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 diagramed in the following table.

Table 5.9-1 - 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 5.9-2, 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 5.9-2 - 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

5.10 Inband Interface - MPEG Data Flow

In S-Mode, the Card/Host Interface supports 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. The bandwidth of the physical interface between the Card and the Host device is increased from 27/38 Mbps to support up to 200 Mbps of aggregate MPEG transport packet data both into and out of the Card simultaneously. 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.

5.11 OOB Interface

The S-CARD device was modified from the [NRSSB]-defined Card to include signaling for the [SCTE55-1] and [SCTE55-2] 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 Card operating in S-Mode or M-Mode includes signaling for the [SCTE55-1] and [SCTE55-2] OOB methods operation, and DSG functionality, where the OOB data can be transmitted either over the extended channel interface or the DSG Resource. Reference Section 9.14 of this document.

Only one method will be active at a time.

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 5.11-1 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 5.11-1 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 5.11-1.

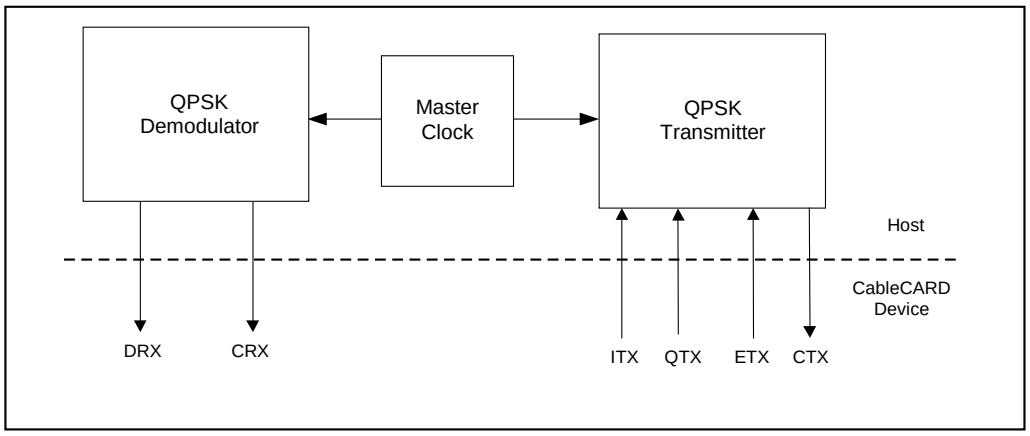


Figure 5.11-1 - CableCARD Out-of-Band Interface

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

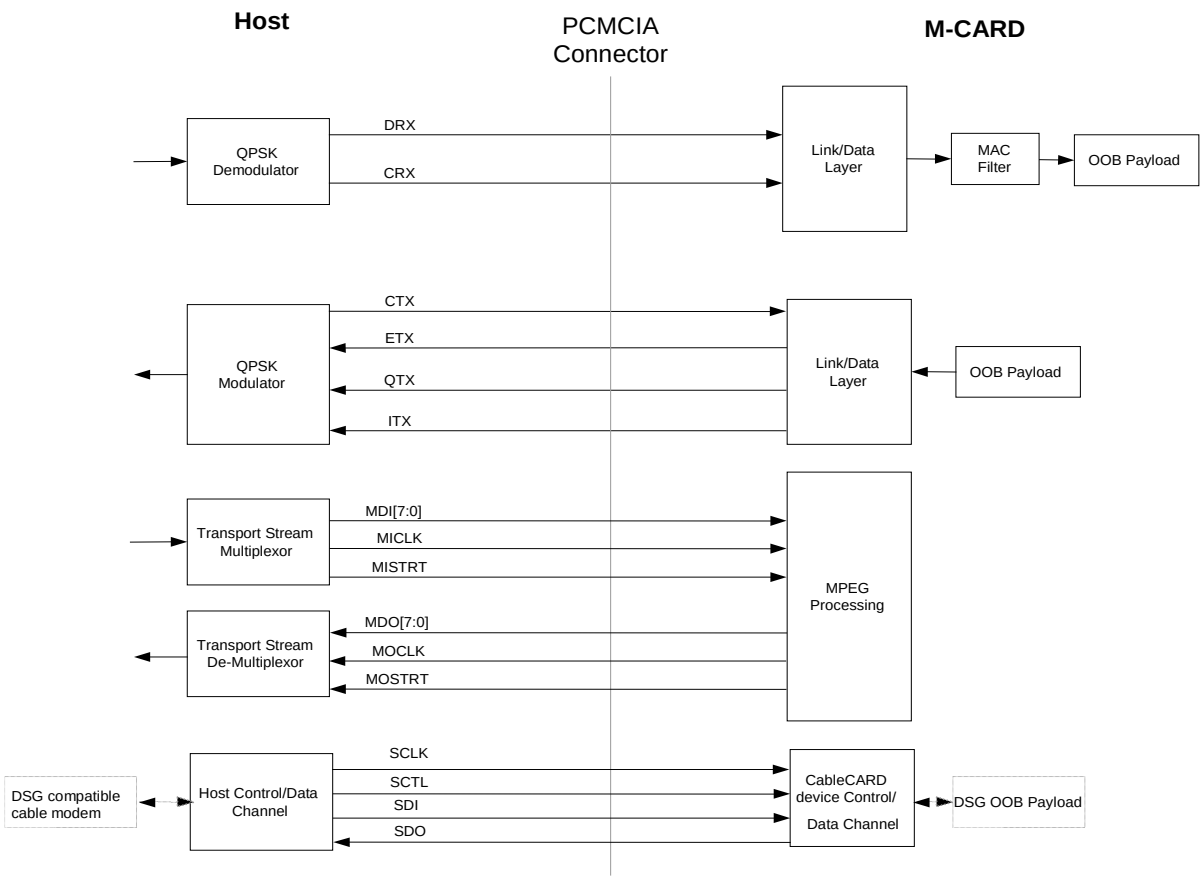


Figure 5.11-2 - M-Mode: CHI Diagram

5.11.1 QPSK

The common modulation method for [SCTE55-1] and [SCTE55-2] 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.

5.11.2 DSG

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

The Host DOCSIS cable modem provides the physical data link and media access control protocols. Unlike the SCTE 55 mode, the data link and media access control protocols for ANSI/SCTE 55-1 and ANSI/SCTE 55-2 are not used. 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 resource. When the Host is operating as an SEB Client, the upstream Conditional Access Messages and network management messages will be transmitted from the Card via IP over the home network interface using the Extended Channel resource.

The interface data rate is:

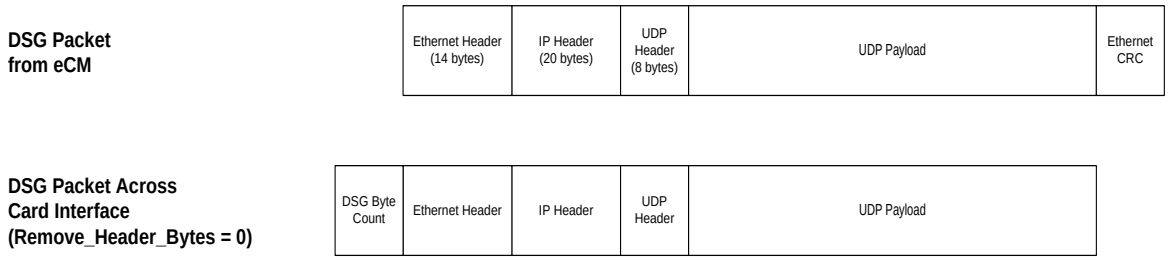
- Downstream direction: 2.048 Mbps

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 encapsulated DSG data. The Card should disregard any invalid packets received from the Host.

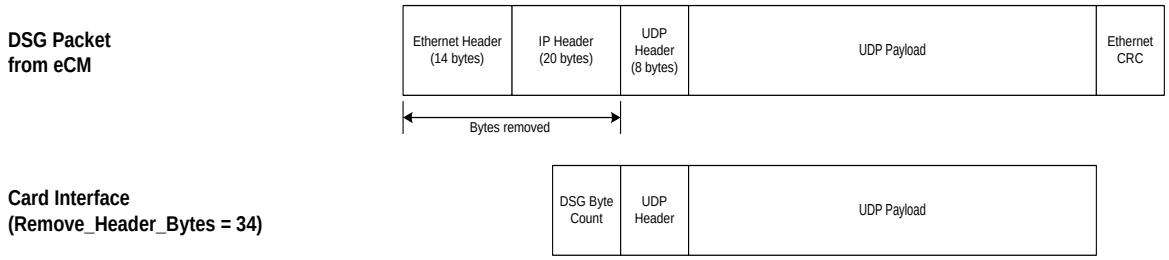
It is the responsibility of the Host to provide buffer space for a minimum of two DSG PDUs, one for transmission to the Card and one for receiving from the DOCSIS channel. **Informational Note:** The DSG rate limits the aggregate data rate to 2.048 Mbps to avoid buffer overflow.

Figure 5.11-3 below shows how the DSG packets are transported across the Card interface with and without removal of header bytes. Prior to transmission across the Card interface, the Ethernet CRC of the DSG packet received from the eCM is removed, then optionally header bytes may be removed in order from the Ethernet header through the IP header and the UDP header, resulting in the removal of X header bytes, where X is designated by the CableCARD as per the remove_header_bytes of the ***set_DSG_mode()*** APDU (note that X may be zero, thus no header bytes are removed). A two-byte field (UIMSBF) containing the DSG byte count of the resulting data payload is prepended to the remaining frame and transmitted across the CHI.

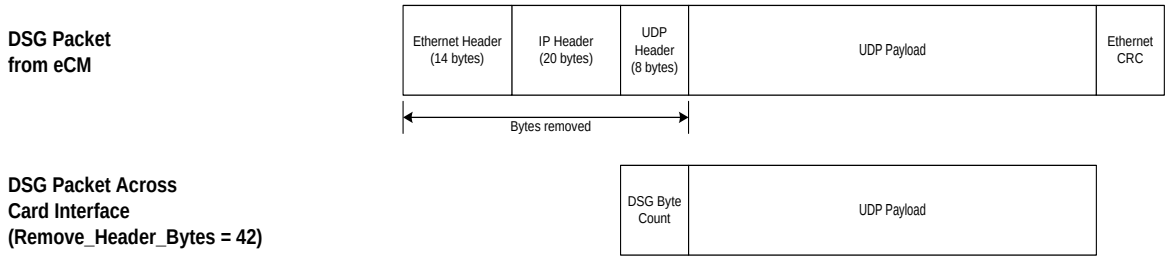
Example 1: Card requests that no bytes be removed from DSG Packet



Example 2: Card requests that Ethernet and IP Header (34 bytes) be removed from DSG Packet



Example 3: Card requests that Ethernet, IP and UDP Header (42 bytes) be removed from DSG Packet



NOTE 1: The IP header may not always be 20 bytes in length, the inclusion of "options" & "padding" will increase the length of the header. It is the responsibility of the Card to determine the number of bytes to remove. The Host must always remove the requested number of header bytes regardless of the size of any individual header.
NOTE 2: DSG packets are not limited to UDP datagrams. UDP datagrams are utilized in these example as informative illustrations only.

Figure 5.11-3 - 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:

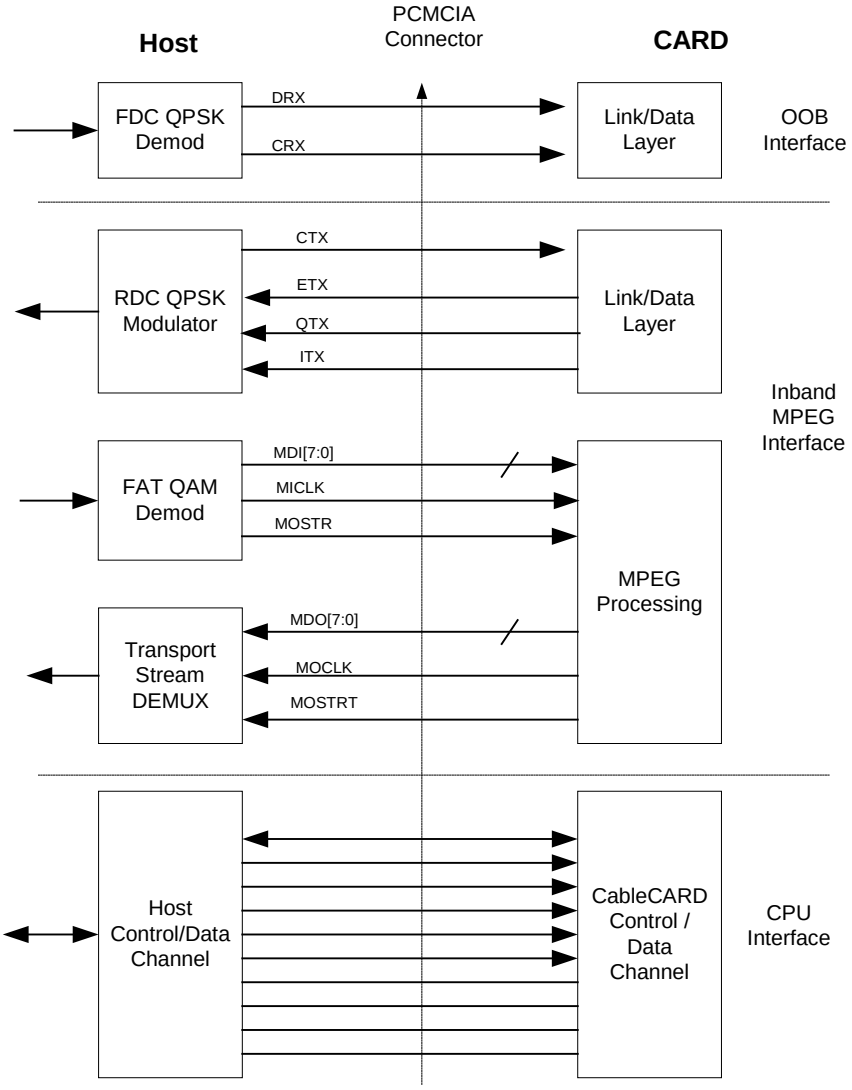


Figure 5.11-4 - S-Mode: CHI Diagram

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 Host senses the presence of the Card and operates as defined in Table 7.1-1. If an M-Host detects that the Card operating mode is S-Mode, it can reject the Card.

The M-Card will start up either directly in M-Mode or follow the PC Card start-up procedure for personality change to S-Mode, depending on the state of VPP1# and VPP2#.

The S-Card, or the M-Card operating in S-Mode, will always follow the PC Card start-up procedure for personality change.

7.1 Interface Pin Assignments

After a PCMCIA reset, the Card operates in memory card mode, with the pin assignments defined in CEA-679C part B, as listed in the left hand column of Table 7.1-1. 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 the Card interface utilizes for the different modes.

Table 7.1-1 - Card Interface Pin Assignments

Pin #	CEA-679C 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

Pin #	CEA-679C 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
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
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, will be decoded during I/O accesses as defined in Features of 16-bit PC Card Asynchronous Interface of [PCMCIA2].

7.2 Deleted, section reserved

7.3 Interface Functional Description

7.3.1 S-Mode Custom Interface

The S-Mode 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).

The Card SHALL present the 16-bit PC Card memory-only interface following the application of VCC or the RESET signal. When the Card is operating in this configuration, its D7-D0 pins 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.

Differences between CEA-679-C part B, PC Card Mode, and the Card when running in S-Mode are identified in Table 7.1-1. The S-Mode column affects A4 to A9 signals, which are now assigned to the OOB RF I/Os, the CE2# signal, which is used to access the Extended Channel, and pins 11, 12, 14, 22, 23, 24, 25, 42, and 57. The MCLKO is provided on pin 14 instead of pin 57 as defined in [NRSSB]. Pin 57 remains the PC Card VS2# signal.

S-Mode SHALL operate in a Host built to compliance with the S-Mode interface signal and I/O assignments as described in Table 7.1-1.

7.3.2 Card Signal Descriptions

7.3.2.1 Power

VCC	If the Card is interfacing to a Host that supports the S-Mode, VCC 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 Card that will be operating in M-Mode is detected, these pins are powered up to 3.3V.
GND	As defined in [PCMCIA2].
VPP1	If the Card is interfacing to a Host that supports the S-Mode, VPP1 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, the VPP1 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, VPP2 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.

7.3.2.2 Sense

CD2::1#	As defined in [PCMCIA2].
VS2::1#	As defined in [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#	The Card detect signal used by the Host to identify if the 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#.

7.3.2.3 PCMCIA Signals

This specification follows [PCMCIA2] Signal Description, Timing Functions, and Electrical Interface descriptions with the following additions:

For the Card in S-Mode, MPEG-2 transport stream interface input and output buses are MDI[7:0] and MDO[7:0]. Signals MCLKI, MCLKO, MISTRT, MIVAL, MOSTRT, MOVAL are used to control the data associated with MPEG-2 transport streams. MCLKI runs at the rate at which bytes are offered to the Card on MDI[7:0]. MCLKO runs at the rate at which bytes are offered by the Card on MDO[7:0]. For Cards which pass the transport stream through, then MCLKO will in most cases be a buffered version of MCLKI with a small delay.

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 provides limits to these timing relationships.

Note: The specification for output timing limits can normally be met by generating the output from the falling edge of MCLKO.

When operating in S-Mode, the Host SHALL provide a continuous MCLKI signal that has a minimum clock period of 110 ns for the MPEG-2 transport stream interface.

In S-Mode, the timing relationship between MCLKO SHALL be derived from MCLKI, and correspondingly the data on MDO[7:0] is the MPEG-2 transport stream interface, delayed and possibly descrambled on MDI[7:0], governed by the rules as described in Section 7.3.5.1.

In S-Mode, MDI[7:0] data and MISTRT SHALL be clocked into the Card on the rising edge of MCLKI.

In S-Mode, MDO[7:0] and MOSTRT SHALL be clocked into the Host on the rising edge of MCLKO.

It is not intended that burst clocking should be employed. Bursty data is handled by the appropriate use of MIVAL and MOVAL.

The MISTRT control signal has the same timing as the MDI[7:0] bus and is valid during the first byte of each transport packet. Figure 7.3-1 defines the timing relationship for the transport stream interface.

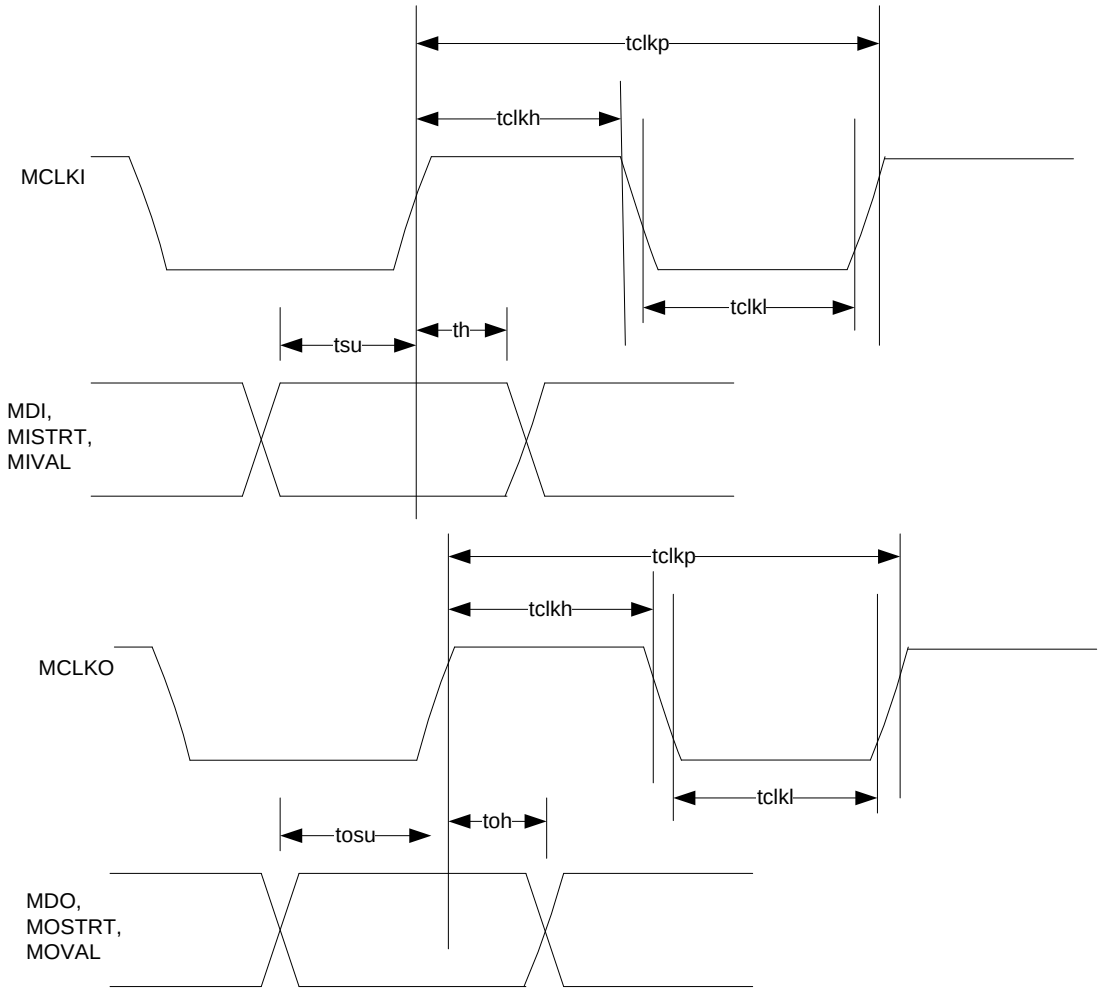


Figure 7.3-1 - Timing Relationships for Transport Stream Interface Signals

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	

When operating in S-Mode, the Card SHALL comply with the minimum relative timing limits on the data signals associated with the MPEG-2 transport stream interface as defined in Table 7.3-1.

The MIVAL signal is only valid in S-Mode and indicates valid data bytes on MDI[7:0]. 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. 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.

MOSTRT is valid during the first byte of an output transport packet. The MOVAL signal, which is also only valid in S-Mode, indicates the validity of bytes on MDO[7:0] in a similar manner as MIVAL. MOVAL may not necessarily be a time-delayed version of MIVAL; see Section 7.3.5.1.

Note: The Host can utilize an interrupt request at the physical layer. Section 7.6.1.1 of this document defines how the PCMCIA IREQ# is available for use by the Host.

The Card in S-Mode responds to an I/O operation by asserting INPACK#; see section 4.4.22 of [PCMCIA2].

7.3.2.4 Card Device Signals

DRX	QPSK receive data input to the Card from the Host.
CRX	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.
ITX	QPSK transmit I-signal output from the Card to the Host and are represented directly to the phase states as defined in [SCTE55-2].
QTX	QPSK transmit Q-signal output from the Card to the Host and are represented directly to the phase states as defined in [SCTE55-2].
ETX	QPSK transmit enable output from the Card to the Host. It is defined to be active high.
CTX	QPSK transmit gapped symbol clock input to the Card from the Host.
MCLKI	MPEG transport stream clock from Host to the Card operating in S-Mode.
MICLK	MPEG transport stream clock from the Host to the Card operating in M-Mode. **Note:** 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.
MISTRT	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.
MDI[7:0]	An 8-bit wide MPEG transport stream input data bus from the Host to the Card.
MCLKO	MPEG transport stream clock from the Card to the Host operating in S-Mode.
MOCLK	MPEG transport stream clock from the Card to the Host operating in M-Mode. MOCLK signal is derived from MICLK and should operate at 27 MHz.
MOSTRT	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. MOSTRT and MDO[7:0] signals are clocked into the Host on the rising edge of MOCLK.
MDO[7:0]	An 8-bit MPEG transport stream output data bus from the Card to the Host.
MIVAL	A control signal that the Card utilizes to indicate valid data bytes on MDI[7:0] when operating in S-Mode.
MOVAL	A control signal that the Card utilizes to indicate the validity of bytes on MDO[7:0] when operating in S-Mode. MOVAL may not necessarily be a time-delayed version of MIVAL.

SCLK	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.
SCTL	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.
SDI	CPU interface serial data from the Host into the Card operating in M-Mode.
SDO	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

The Host SHALL support QPSK transmit I-signal (ITX) rates of 772/1544 and 128 Ksymbol/s (Tx I channel).

The Host SHALL support QPSK transmit Q-signal (QTX) rates of 772/1544 and 128 Ksymbol/s (Tx Q channel).

The Host SHALL NOT transmit the values of ITX and QTX when ETX is inactive.

7.3.2.5 M-Mode Device Signals

When an M-Card is inserted into an M-Host, it can start up directly in M-Mode when the operating mode state is VPP1# Low while VPP2# is High.

The Card operating in M-Mode SHALL comply with the M-Mode signal pin assignments as described in Table 7.1-1.

When operating in M-Mode, the Host SHALL provide a continuous MICLK signal for the MPEG-2 transport stream interface that operates at 27Mhz.

When operating in M-Mode, the timing relationship between MOCLK SHALL be derived by the Card's MICLK.

- MDI[7:0] data and MISTRT SHALL be clocked into the Card on the rising edge of MICLK.
- MDO[7:0] and MOSTRT SHALL be clocked into the Host on the rising edge of MOCLK.

7.3.3 Card Type Identification

The CableCard interface will be detected before the socket notifies Card Services of an insertion event. To initially power up a PC Card and determine its characteristics, VCC and VPP/VCORE must be at a voltage indicated by the Voltage Sense pins as defined in [PCMCIA2].

The Host SHALL utilize the Card type detection mechanism to determine the Card interface and initial voltage requirements as defined in section 3 of [PCMCIA2].

The Host SHALL use the CD[2::1]# signals to detect when the Card is inserted.

Note: Hosts need not support the detection mechanisms for CardBus PC Cards, but may optionally do so as defined by [PCMCIA2].

7.3.3.1 S-Mode

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.). When the Card is operating in S-Mode, its CIS SHALL include a custom interface subtuple (CCST_CIF) that provides the interface ID number (STCI_IFN) defined by PCMCIA (0x341). In PCMCIA memory mode, the CableCARD interface is a PC Card Custom Interface that reads the Card Information Structure (CIS) on the even addresses (first byte at address 0x 000, second byte at address 0x 002, etc.)

The Card CIS also includes the field name of the product of subtuple TPLLV1_INFO defined as "OPENCABLE_POD_MODULE" in the tuple CISTPL_VER_1. This information in the CIS is required to ensure backup operation in case of trouble when the CI stack is lost (e.g., power shut down, Card extraction).

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

7.3.3.1.1 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 Card Information Structure (S-Mode Only)

In 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 characteristics of Function Configuration registers 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 the Card has been installed.

In S-Mode, the Card SHALL implement the Card Configuration table in its Card Information Structure (CIS) as defined in [PCMCIA2].

In S-Mode, the Card SHALL provide the set of tuples in the CIS as defined in Table 7.3-3.

Table 7.3-3 - 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 the Card's S-Mode Attribute and Configuration Registers can be found in Annex D of this document.

After S-Mode Card Personality Change, the CIS SHALL no longer be available.

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

7.3.4.1 Operation After Card Personality Change

The Card in S-Mode starts up as a PC Card in memory only mode, then goes through a personality change according to the PC Card standard.

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.

7.3.4.2 M-Mode Detection

In M-Mode, the Host SHALL determine the physical interface before applying power to the VCC and VPP pins on the Card/Host Interface.

The M-Card SHALL be designed such that VS1# = ground and VS2# is connected directly to MDET.

Hosts capable of operating in M-Mode SHALL follow these steps:

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 described in Figure 7.3-2 and the Card and Host Pull-ups and Pull-downs Table 7.4-6 in Section 7.4.1.3.
3. Detect the presence of the Card using CD1# and CD2# as described in Card type detect Host signaling operation.
4. High Z on VCC pins until after the Card device type is determined.
5. Determine if VS2# is tied to MDET. If VS2# is tied to MDET, the Host proceeds with its power up in M-Mode.
6. Apply power to the VCC pins (3.3V) and VPP2 (5.0V) once the Host determines that the Card is in M-Mode. The VPP2 pin is provided to support the Card that contains an optional smart card.

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

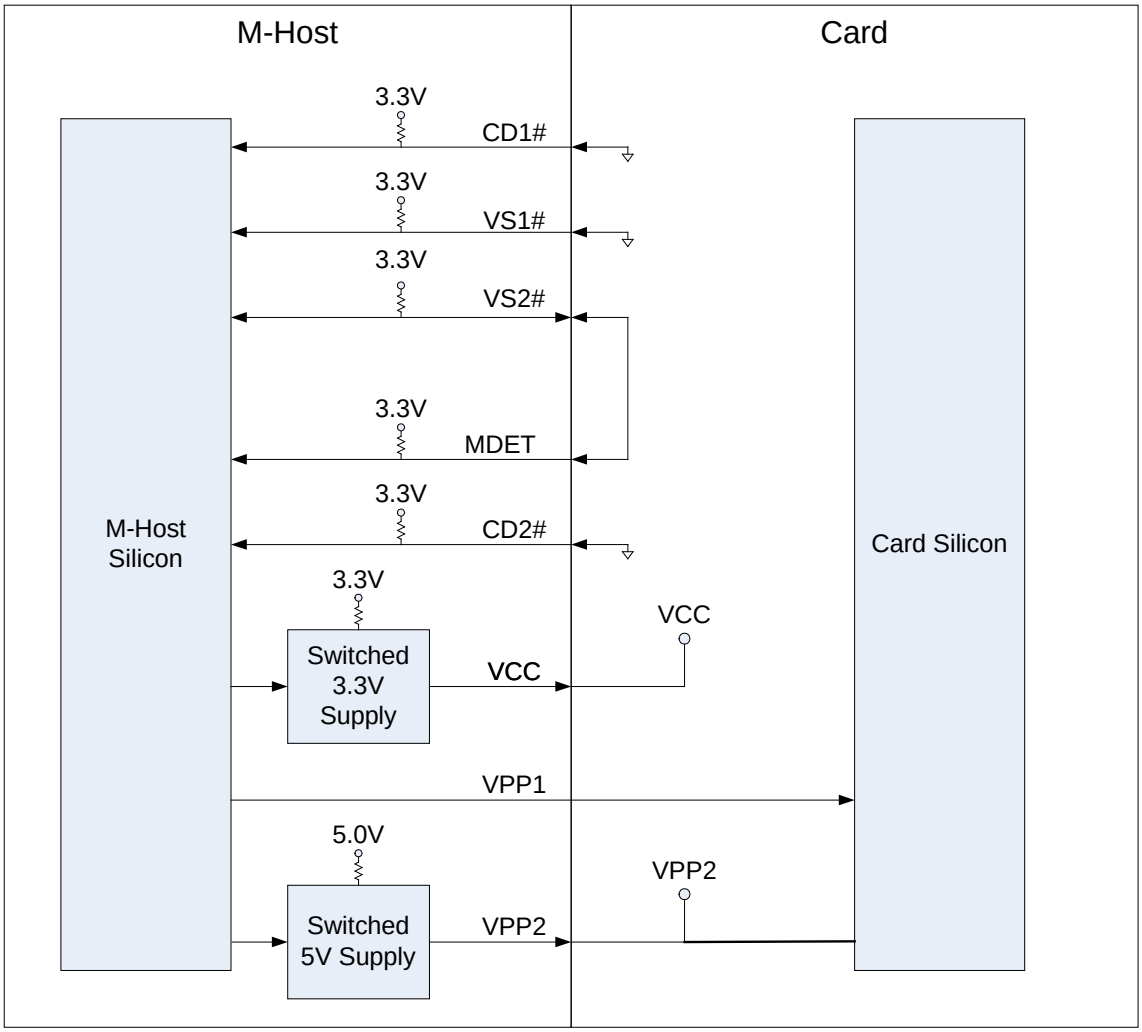


Figure 7.3-2 - Card Type Detection Signals

Hosts capable of operating in M-Mode can be tested by toggling VS2# and watching to see that MDET tracks VS2#. If the M-Host does not support S-Mode on the Card/Host Interface, the Host may choose to not power the PC Card and should display an error message indicating the Card inserted is not supported as defined in Annex B of this document.

Hosts capable of operating in M-Mode are not required to support PC Card memory-only interface; they MAY immediately operate in the M-Mode.

As power is applied, all interface pins on the M-Card SHALL be defined in a manner that will not contend with the Host until the Card knows in the operating mode. 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 pin is at a logic low while the VPP2 pin is at a logic high (5V), the Card SHALL initialize to M-Mode.

If the VPP1 and VPP2 pins are at a logic high, then the Card SHALL initialize to S-Mode.

When VPP1/VPP2 configuration is set to one of the two reserved settings in Table 7.3-4, the M-Card SHALL keep SDO (pin 60) and READY (pin 16) low. If SDO and READY have not gone active after 5 seconds, the Host may use this as an indication that the M-Card is not configured correctly to respond.

Note: When the M-Card is operating in S-Mode, only the READY line has significance, and SDO has no meaning.

The following table summarizes VPP1/VPP2 pin configurations, and the associated Card configuration operating mode.

Table 7.3-4 - 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

7.3.5 MPEG Transport Interface

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 MPEG-2 multiplexed packets are sent over this transport stream interface and are 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].

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. This 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.

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.

The MPEG interface consists of an input clock (MCLKI for S-Mode and MICLK 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.

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.

7.3.5.1 S-Mode

The Card/Host Interface supports independent physical connections for the transport stream and for commands. The returned transport stream may have some of the incoming transport packets returned in a descrambled form.

The Card SHALL accept an MPEG-2 Transport Stream that complies with [ISO13818-9], consisting of a sequence of transport packets, either contiguously or separated by null data.

The Host SHALL provide an MPEG-2 Transport Stream that complies with [ISO13818-9], consisting of a sequence of transport packets, either contiguously or separated by null data.

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:

tmdvmax = (n * TMCLKI) + (2 * TMCLKO).

And-

tmdvmax <= 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.

Note: 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.

The Card SHALL support a byte transfer clock (MCLKI) period of 110 ns. minimum

The Host SHALL provide a byte transfer clock (MCLKO) period of 110 ns minimum.

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.

An S-Card or the M-Card operating in S-Mode SHALL 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.

7.3.5.2 M-Mode

The Card MPEG Packet (CMP) consists of a 188-byte MPEG packet with a pre-pended 12-byte header. In M-Mode, the Host SHALL transmit each CMP packet header and MPEG packet with no gaps after the packet start signal is active. In M-Mode, the Host SHALL transmit MPEG packets that consist of 200 contiguous bytes including the 12-byte header.

In M-Mode, the Card will keep track of the number of bytes received, starting with the packet start signal, and form a packet from the first 200 bytes.

In M-Mode, the Card SHALL provide a delay variation (jitter) of no more than one MICLK period for each CMP packet header and MPEG.

In M-Mode, after the Card has received the first 200 bytes, it SHALL discard any following bytes 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. 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.

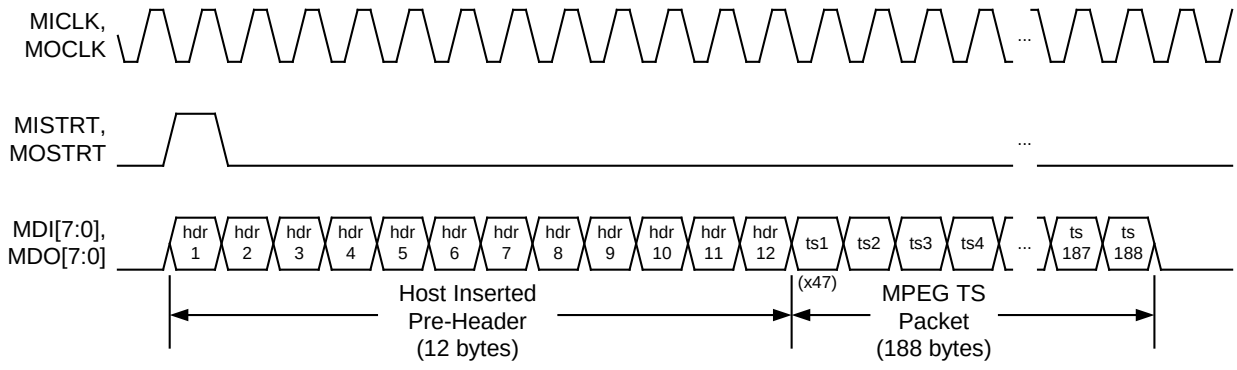


Figure 7.3-3 - CMP Diagram

7.3.5.2.1 M-Mode MPEG Transport Stream Pre-Header

In M-Mode, the MPEG Transport Stream pre-header SHALL include a 12-byte field, as described in Figure 7.3-4, pre-pended by the Host to each MPEG packet sent across the Card Host Interface. This pre-header provides identification information to allow packets from multiple transport streams to be multiplexed prior to delivery to the Card.

The Card SHALL NOT modify the LTSID field in the M-Mode MPEG Transport Stream pre-header. The data format is shown in Figure 7.3-4. The transport streams are identified by local transport stream IDs (LTSID), which the Host is responsible for generating. 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.

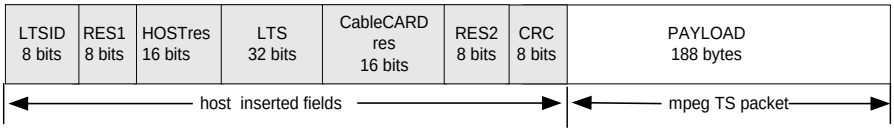


Figure 7.3-4 - M-Mode MPEG Transport Stream Pre-Header

LTSID Local Transport Stream ID - All packets in a given transport stream are tagged with the same unique LTSID. This allows 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.

Res1, Res2 Two 8-bit fields reserved for future use with a default value of 0x00.

Host_reserved 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.

LTS Local Time Stamp - A 32-bit local time stamp, whose value is set by the Host. Setting a value in this field is optional for the Host.

CableCARD_reserved A 16-bit field. The default value is 0x00. The usage of this field is optional for the Card.

CRC 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. 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.

In M-Mode, the Host SHALL tag all packets in a given transport stream with the same unique LTSID.

In M-Mode, the Card SHALL ignore the MPEG transport pre-header Res1/Res2 bit fields when the default value for both is 0x00.

In M-Mode, the Host SHALL ignore the MPEG transport pre-header Res1/Res2 fields when the default value for both is 0x00.

In M-Mode, the Card SHALL NOT modify the values in the Host_reserved fields.

In M-Mode, the Card SHALL NOT modify the Local Time Stamp (LTS) value.

Setting a value in the LTS field is optional for the Host. If the Host chooses to set the LTS value, it SHALL use 32-bit value.

The Host MAY use the local time stamp to manage MPEG timing of the packets received from the Card.

In M-Mode, the Host SHALL insert an 8-bit Cyclic Redundancy Check (CRC) across the 11 bytes of the pre-header as the means of calculating that the LTSID, the CableCARD_reserved, Host_reserved, and LTS have been transferred across the transport interface without errors.

In M-Mode, if the Card inserts data into the CableCARD_reserved field, it SHALL recalculate the CRC for those packets.

7.4 Electrical Specifications

In order to remain compliant with the PC Card standard [PCMCIA2], regardless of the powering state of the Host (i.e., active or standby), the Host and the Card should implement the power characteristics as defined in this document.

A Common Interface module is implemented as a variant of the 16-bit PC Card Electrical Interface 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 the Card is first inserted into the Host, before configuration, it 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.
3. CE2# SHALL be ignored and interpreted by the Card as a logic high.
4. 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).
5. Signals BVD1 and BVD2 SHALL remain logic high.

7.4.1 DC Characteristics

7.4.1.1 S-Mode

In order to remain compliant with the PC Card standard [PCMCIA2], regardless of the powering state of the Host (i.e., active or standby), the following power management features are required.

- If the Host supports S-Mode, it SHALL permanently supply 3.3V on the VCC pins (Card Inserted/Removed) with the capability of supplying up to a maximum of 1 amp total on the VCC pins (500 mA each) at 3.3 VDC per Card supported.
- If the Host supports S-Mode, it SHALL supply 5V on the VPP pins when requested by the Card's CIS with the capability of supplying up to 250 mA total on the VPP pins (125 mA each) at 5 VDC per Card supported.
- In S-Mode, when the Card is inserted, the Host SHALL supply 5 V on the VPP pins if requested by the Card's CIS.

  Otherwise, the Host continues to supply 3.3 V on the VPP pins while the Card is installed.
- In S-Mode, upon removal of the Card, the Host SHALL revert to or continue to supply 3.3V on the VPP pins.

- If the Host does not support the value of 0x03 in the Power Field of Feature Selection Byte (TPCE_FS) it receives from the Card, then it SHALL continue to supply a nominal voltage of +3.3V to both the VPP1 and VPP2 pins.

  Note: If the Host supports S-Mode and receives a value of 0x3 in the Power field of the Feature Selection Byte (TPCE_FS) from the Card, it is not required to support the separate nominal voltage parameter descriptors in the power descriptor structures for VPP1 and/or VPP2.

- In S-Mode, the Card SHALL only support the value of 0x02 in the Power field of the Feature Selection Byte (TPCE_FS) when the Card requires a switched nominal voltage level of +5V on the VPP lines according to [PCMCIA4].
- In S-Mode, the Card SHALL NOT draw more than 2.5 watts averaged over a period of 10 seconds.
- If the Host supports S-Mode, the OOB Receive circuitry SHALL continue to operate in all powering states of the Host.
- If the Host supports S-Mode, it SHALL support hot insertion and removal of the Card.
- In S-Mode, the Card SHALL implement the mechanical Low Voltage Keying as defined in [PCMCIA2].
- In S-Mode, 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.
- In S-Mode, the Card SHALL support 3.3V hot insertion.

Note: The Card does not have to meet the requirement 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

- Hosts capable of operating in M-Mode SHALL provide 3.3 V on VCC and 5V on VPP2 with the DC Characteristics as defined in Table 7.4-1.
- Hosts capable of operating in M-Mode SHALL be capable of providing up to 1 Amp total on the VCC pins (500 mA each) per Card supported.
- Hosts capable of operating in M-Mode SHALL be capable of providing up to 125 mA on the VPP2 pin.
- In M-Mode, the Card SHALL NOT draw more than 1.5 watts averaged over a period of 10 seconds.
- Pin type LogicPC - DC Signal levels for 3.3V Signaling.
- Hosts capable of operating in M-Mode SHALL continue to operate their OOB circuitry in all powering states of the Host.
- Hosts capable of operating in M-Mode SHALL detect the state of the CD1# and CD2# pins and power-down the VCC and VPP interface pins upon Card removal, and upon Card insertion, determine the Card type prior to applying VCC and VPP power.
- Hosts capable of operating in M-Mode SHALL tolerate voltage ranges on VCC (3.0 to 3.6V) at up to 1 Amp total, and VPP2 (4.75 to 5.25V) with maximum supply current of 0.125 A as defined in Table 7.4-1.

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_{CC}	Supply Current	-	1.0	A	2
VPP2	Supply Voltage	4.75	5.25	V	1
I_{PP2}	Supply Current	0	0.125	A	2
Notes: 1. There is no standby power mode for the Card. 2. Hosts capable of operating in M-Mode provide 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 support the Electrical Interface as defined in [PCMCIA2]. Support by Hosts for overlapping I/O address windows as defined in [PCMCIA2] is optional.

In S-Mode, Cards SHALL use an independent I/O address window 4 bytes in size.

In S-Mode, the Host SHALL detect Card insertion and removal using CD1# and CD2#.

In S-Mode, the Card SHALL provide a means of allowing the Host to detect Card insertion and removal using CD1# and CD2# as described in [PCMCIA2].

In S-Mode, the Card SHALL NOT implement the battery voltage detect (BVD[2::1]) function.

In S-Mode, the Card SHALL implement the I/O transfers to the Host for 8-bit read and write timing as defined in Figure 4-7 and Figure 4-8 of [PCMCIA2].

In S-Mode, the Host SHALL implement the I/O transfers to the Card for 8-bit read and write timing as defined in Figure 4-7 and Figure 4-8 of [PCMCIA2].

In S-Mode, the Card SHALL only use 8-bit read and write modes.

7.4.1.3.2 Card Signal Types

Table 7.4-2 summarizes what Card signal types are used for each of the interface signals and operating modes.

The Card SHALL utilize the signal types for each of the designated pin assignments as defined in Table 7.4-2.

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, MICLK	CCLK
A[25:17]	MDI[7:0], MISTRT	MDI[7:0], MISTRT	LogicCB (M-Mode); LogicPC (S-Mode)
D[15:8], BVD1	MDO[7:0], MOSTRT	MDO[7:0], MOSTRT	LogicCB (M-Mode); LogicPC (S-Mode)
A[9:4]	DRX, CRX, CTX, ITX, QTX, ETX	DRX, CRX, CTX, ITX, QTX, ETX	LogicPC

PC Card Memory-Only Signals	S-Mode Signals	M-Mode Signals	Signal Type
A[2:0],RFU	Unused, A[1:0], INPACK#	SCTL, SCLK, SDI, SDO	LogicPC
VS1#, VS2#	VS1#, VS2#	VS1#, VS2#	Sense[1]
RESET	RESET	RESET	LogicPC
WP	IOIS16#	MDET	LogicPC[1]
VPP1, VPP2	VPP1, VPP2	VPP1, VPP2	VPP1, VPP2
CD[2:1]#	CD[2:1]#	CD[2:1]#	Sense

Notes:
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 voltage characteristics described in Section 7.4.
3. When the Card is operating in S-Mode, MICLK (A12 pin 21) will have the characteristics of CCLK.

Table 7.4-3 provides reference for the Card on DC signaling levels associated with signal type.

The Card SHALL support the DC Signal levels for each signal type as summarized in Table 7.4-3, defined in [PCMCIA2].

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 shows the DC signaling characteristics for the “LogicPC” signaling level.

The Card SHALL support "LogicPC" signaling level parameters as defined in Table 7.4-4.

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_{IH}		2.0	VCC + 0.3	V	
V_{IL}		-0.3	0.8	V	
V_{OH}		2.4 (VCC-0.2)		V	1
V_{OL}			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 shows the DC signaling characteristics for the "LogicCB" and "CCLK" signaling level.

The Card SHALL support "LogicCB" signaling level parameters as defined in Table 7.4-5.

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_{IH}			0.475VCC	VCC + 0.5	V	
V_{IL}			-0.5	0.325VCC	V	
I_{IL}		0 < Vin < VCC		+/-10	uA	
V_{OH}		Iout = -150uA	0.9VCC		V	
V_{OL}		Iout = 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 required Pull-up and Pull-down resistance values that satisfy signal switching impedances based upon the mode of operation.

The Card SHALL support pull-up/pull-down resistance requirements on the signal pins defined in Table 7.4-6 while meeting all AC timing requirements.

The Host SHALL support pull-up resistance requirements depending on its operational mode, on the signal pins defined in Table 7.4-6, while meeting all AC timing requirements.

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.>= 10K	pullup to Host 3.3V R.>= 10K	6
Voltage Sense	VS[2:1]#			pullup to Host 3.3V 10K <= R.<= 100K	pullup to Host 3.3V 10K <= R.<= 100K	
Card Type Detect	MDET				pullup to Host 3.3V R >= 100K	
Control Signal	RESET	pullup to VCC R >= 100K	pullup to VCC R >= 100K			3
MPEG Interface	MICLK, MISTRT, MDI[7:0]		pulldown R >= 100K			1
	MOCLK		pulldown R >= 100K			1
	MOSTRT				Pullup to Host 3.3V R.>= 10K	1
	MDO[7:0]	pulldown R >= 100K	pulldown R >= 100K			1
OOB Interface	CRX, DRX, CTX	pulldown R >= 100K	pulldown R >= 100K			1, 5
	ITX, QTX, ETX	pulldown R >= 100K	pulldown R >= 100K			1, 4
CPU Interface	SCTL, SCLK, SDI		pulldown R >= 100K			1, 5
	SDO			pullup to Host 3.3V R.>= 10K		2, 4

Notes:

1. Due to PC Card Requirement for pull-ups and pull-downs on the Memory Interface.
2. S-Host has R >= 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 an AC requirements of this signal.
3. Note 3 of Table 4-16 in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
4. Note 4 of Table 4-16 in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
5. Note 5 of Table 4-16 in Section 4.7.1 of [PCMCIA2] Release 8.0 applies.
6. The CableCARD device forces 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 and M-Mode Signal Parameters

The Card SHALL comply with the OOB FDC signal parameters and RDC timing requirements as defined in Table 7.4-7 and illustrated in CableCARD Device Output and Input Timing Diagrams in Figure 7.4-1 and Figure 7.4-2.

Note: Signal Parameters are 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].

The Host SHALL support signal voltage levels that are compatible with normal 3.3V CMOS levels.

The Host SHALL support CTX (Transmit Gapped Symbol Clock) for 772/1544 and 128KHz.

Table 7.4-7 - S-Mode/M-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	129			Notes 1, 2, 3
Clock Low Time (T_{LOW})	CTX, CRX	ns	129			Notes 1, 2, 3
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. 3. Minimum value derived assuming a duty cycle of 60/40.						

The Host SHALL support CRX to maximum frequency equal to 3088 KHz.

The Host SHALL comply with CTX high time (THCTX) timing limit of 129 ns minimum.

The Host SHALL comply with CTX low time (TLCTX) timing limit of 129 ns minimum.

The Host SHALL comply with the minimum ETX,ITX,QTX signal delay time (t_D) of 5 ns up to 180 ns maximum delay, measured from CTX rising at 1.5V to ETX, ITX, QTX going valid/invalid at 1.5V.

The Host SHALL comply with the minimum DRX signal set-up time (T_{su}) requirement of 10 ns from time signal reaches 90% of high level (rising) or 10% of high level (falling) until CRX mid-point transition.

The Host SHALL comply with the minimum DRX signal hold time (T_h) of 5 ns from CRX mid-point until signal reaches 10% of high level (rising) or 90% of high level (falling).

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.

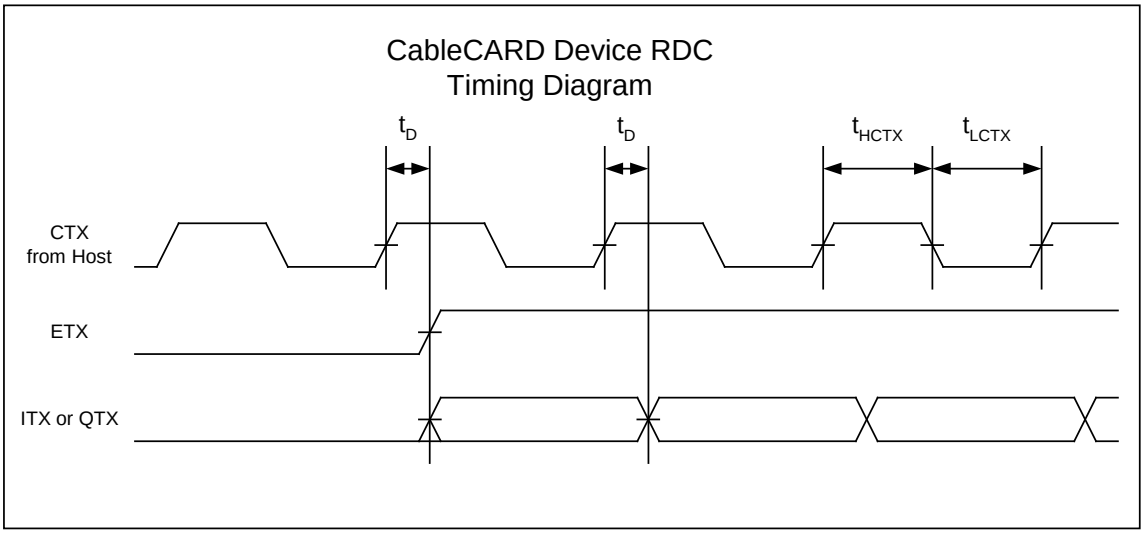


Figure 7.4-1 - CableCARD Device Output Timing Diagram

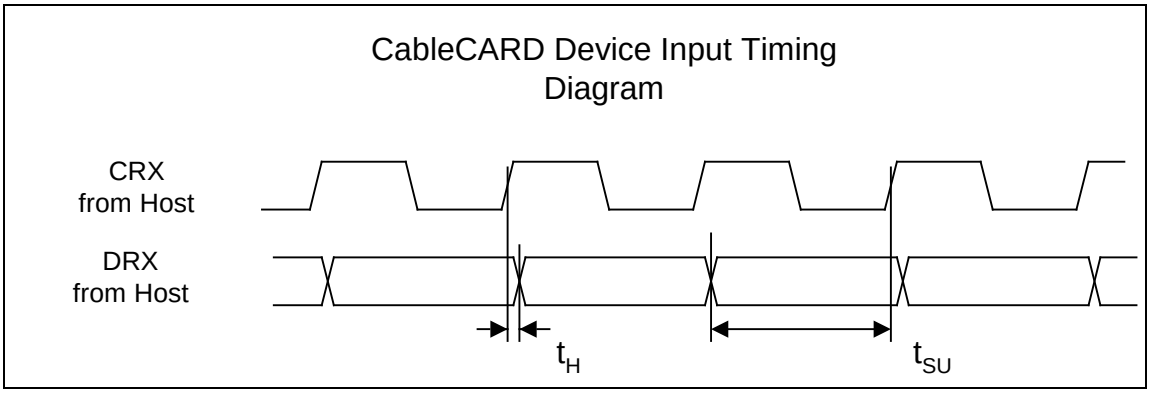


Figure 7.4-2 - CableCARD Device Input Timing Diagram

7.4.2.2 M-Mode

When the Card is operating in M-Mode, the AC signaling characteristics of CCLK, LogicPC and LogicCB SHALL be as 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 MICLK 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.

In M-Mode, the Card SHALL comply with the Power-On and Reset Timing Requirements defined in Table 7.4-8 and illustrated in Figure 7.4-3.

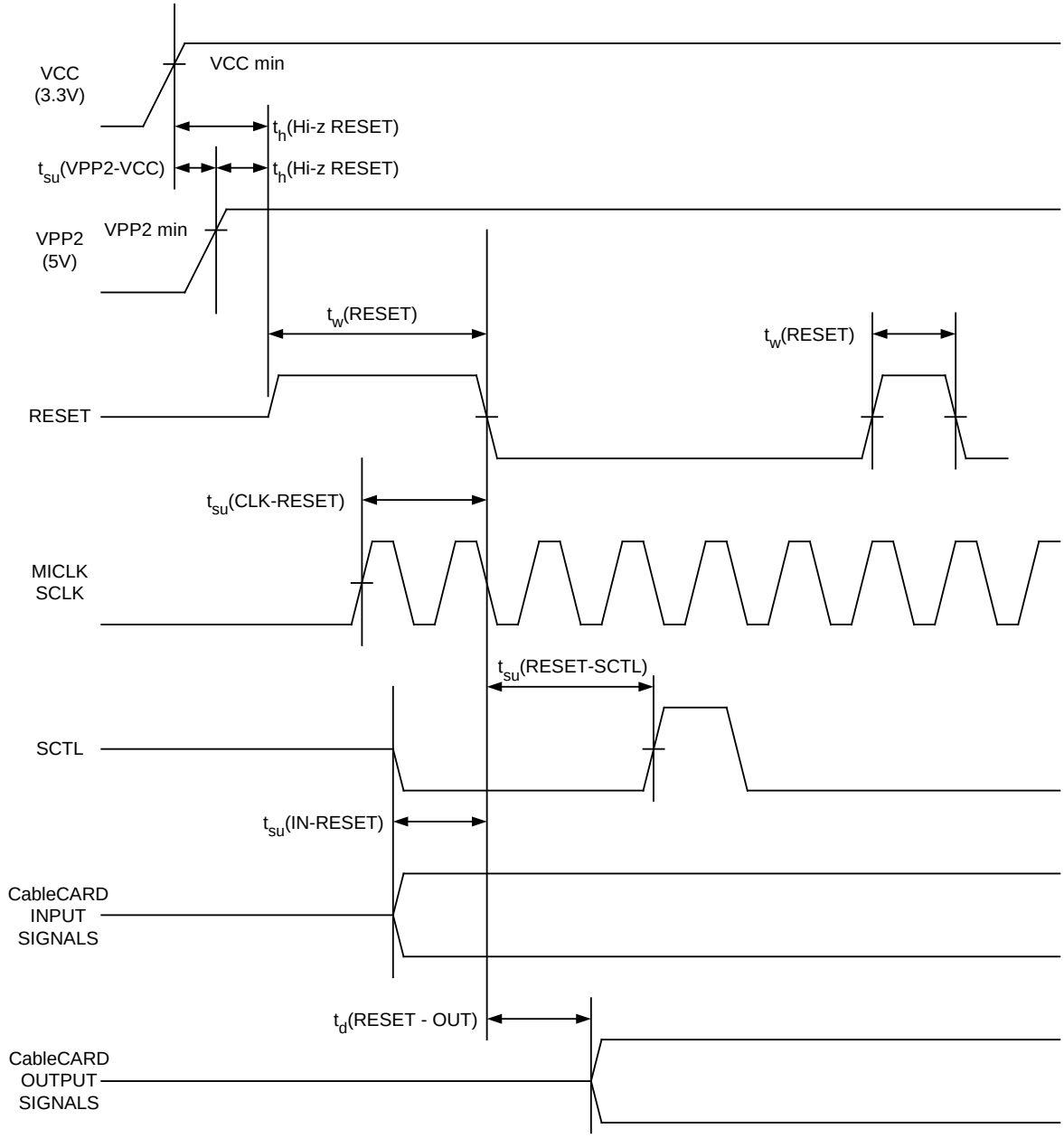


Figure 7.4-3 - M-CARD Power-On and Reset Timing Diagram

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 should not exceed VCC=0.3V until VCC has reached VCC min as defined in Table 7.4-1.
2. In M-Mode, the Card input logic signals are Hi-z from when VCC and VPP2 are applied until RESET is asserted.
3. In M-Mode, the 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.

MPEG packets are sent across the interface from the Host to the Card and returned to the Host with a delay variant from receipt to transmission back to the Host.

The delay variation through the Card is the MPEG transport interface input clock, i.e., ± 1 MICLK 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 Card SHALL support the M-Mode MPEG transport Timing requirements as defined in Table 7.4-9.

The Host SHALL comply with the M-Mode MPEG transport Timing requirements defined in Table 7.4-9.

Table 7.4-9 - M-CARD MPEG Transport Timing

Symbol	Parameter	Min	Max	Units	Notes
t_{cyc}	MICLK and MOCLK Cycle Time	37.00	37.074	ns	1, 2, 3
t_{high}	MICLK and MOCLK High Time	15		ns	3
t_{low}	MICLK and MOCLK Low Time	15		ns	3
t_{su}	Input Setup time of MDI[7:0] and MISTRT to MICLK and Input Setup Time of MDO[7:0] and MOSTRT to MOCLK	7		ns	4
t_h	Input Hold time of MICLK to MDI[7:0] and MISTRT and Input Hold Time of MOCLK to MDO[7:0] and MOSTRT	0		ns	4
t_{val}	MICLK to and MOCLK to Output Signal Valid Delay.	2	18	ns	4

Notes:

1. MOCLK will be derived directly from MICLK.
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 CPU Interface Timing

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

In M-Mode, CPU functions are determined by the following control signals and the timing relationship between the signals.

Hosts capable of operating in M-Mode SHALL provide their serial clock (SCLK) at a nominal rate of 6.75 MHz (27 MHz/4).

In M-Mode, the Card SHALL change its serial control signal on the falling edge of the SCLK signal.

Hosts capable of operating in M-Mode SHALL change their serial data input (SDI) on the falling edge of the SCLK signal. In M-Mode, the Card SHALL input its serial data input (SDI) on the rising edge of the SCLK signal.

In M-Mode, the Card SHALL change its serial data output (SDO) on the falling edge of the SCLK signal. Hosts capable of operating in M-Mode SHALL input their serial data output (SDO) data on the rising edge of the SCLK signal.

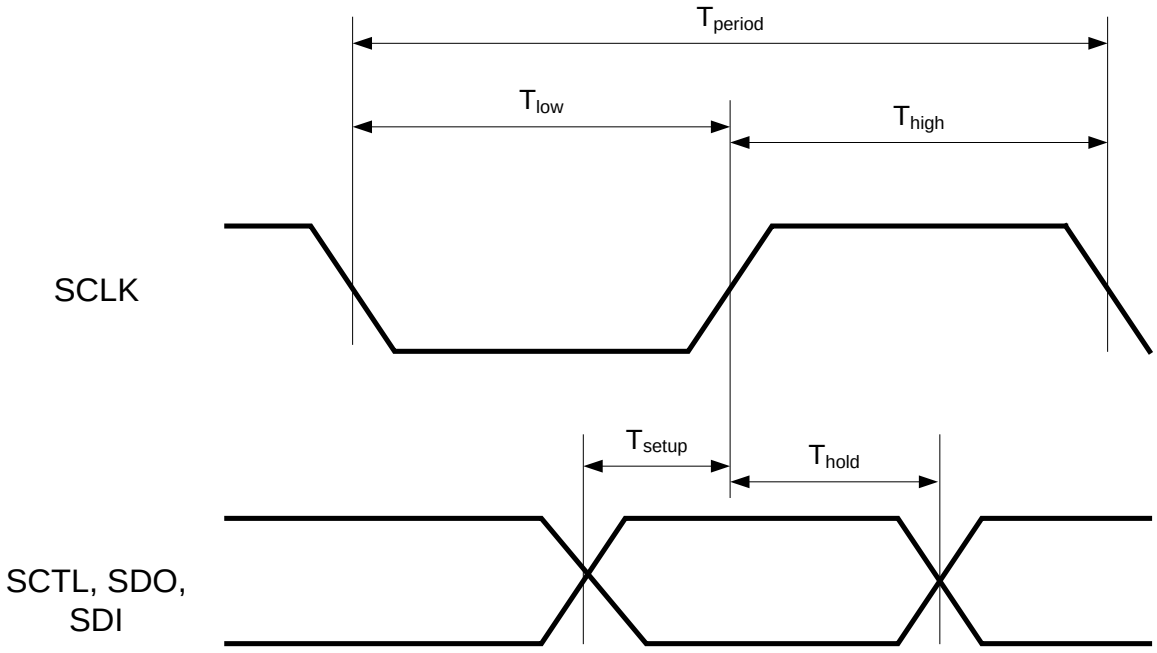


Figure 7.4-4 - M-Mode Serial Interface Timing Diagram

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_{high}	74	88	59	ns
	T_{low}	74	88	59	ns
SCTL	T_{setup}	n/a	n/a	7	ns
	T_{hold}	n/a	n/a	0	ns
SDI	T_{setup}	n/a	n/a	7	ns
	T_{hold}	n/a	n/a	0	ns
SDO	T_{setup}	n/a	n/a	7	ns
	T_{hold}	n/a	n/a	0	ns
Note: Minimum times are specified with 0 pF equivalent load; maximum times are specified with 30 pF equivalent load.					

In M-Mode, the Card SHALL support the Serial Interface Timing requirements as defined in Table 7.4-10.

Hosts capable of operating in M-Mode SHALL comply with the Serial Interface Timing requirements as defined in Table 7.4-10.

Hosts capable of operating in M-Mode SHALL support the duty cycles on SCLK of no less than 40%, no more than 60%.

7.5 Mechanical Specifications

7.5.1 Form Factor

The mechanical design of the Card SHALL comply with the PC Card and CardBus specifications as defined 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 of Low Voltage Keying as defined in [PCMCIA3], with the exception that the length dimension can range from 85 mm to 102 mm.

The Card SHALL include the CardBus/CardBay PC Card Recommended Connector Grounding as defined in [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 as 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 with the Card inserted 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 comply to PC Card guidance standard as defined in [PCMCIA3].

7.5.5 Grounding/EMI Clips

The Card SHALL provide contact resistance grounding and EMI clips as defined in [PCMCIA3].

7.5.6 Connector Reliability

The Host SHALL meet all connector reliability test requirements as defined in [PCMCIA3].

7.5.7 Connector Durability

The Host SHALL meet all connector durability test requirements as defined in [PCMCIA3].

7.5.8 PC Card Environmental

The Card SHALL meet all environmental test requirements for Environmental Resistance as defined in [PCMCIA3]. The Card SHALL operate at temperature specification of up to 55ºC, as defined in [PCMCIA2]. This is primarily to enable reliable battery operation in Cards.

The Host SHALL limit the temperature rise between the ambient environment outside the Host and the ambient environment surrounding the Card to 15ºC when the Card is dissipating its full rated power.

Notes:

To minimize the risk of Card/Host Interface misoperation, Cards containing batteries should follow the recommendations in the PC Card standard on battery placement.

Host designers should conform to the relevant mandatory safety specifications with regard to Host thermal design and 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 distinct communication paths, the Data Channel and the Extended Channel, share the same pins on the PC Card connector (see Table 7.1-1).

Except for the physical layer, the Data Channel and Extended Channel are logically separate from each other, each containing its own buffers and packet queue. No operation at the transport, link, session or application layer of the Data Channel inhibits reception or transmission of data on the Extended Channel, and no operation at the link layer of the Extended Channel inhibits reception or transmission of data on the Data Channel.

At the physical layer, the Card SHOULD NOT expect the Host to perform a write operation on the Data or Extended Channel in any order or preference, i.e., these channels can be written to at will by the Host whenever the corresponding Status Register FR (Free) bit is set to "1".

At the physical layer, the Host SHOULD NOT expect the Card to set the Status Register DA (Data Available) bit on the Data or Extended Channel in any order or preference, i.e., when the Status Register DA bit is set to "1" on the Card Data or Extended Channel, the Host reads the data for that channel regardless of the state of the Status Register DA bit on the other channel.

Data Channel Operation - 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 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 data is available. 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.

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 Table 7.6-1.

Only two address lines, A0 and A1, are decoded by the interface. This block of 4 bytes can be placed anywhere within its own address space by suitable decoding or mapping of other address lines within the Host.

Table 7.6-1 - Hardware Interface Registers

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

The Status Register is depicted in Table 7.6-2.

Table 7.6-2 - Status Register

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

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.

IIR (Initialize Interface Request) is sent by the Card to request the Host to perform a Card reset.

R indicates reserved bits that are set to zero.

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

The Control Register is depicted in Table 7.6-3.

Table 7.6-3 - Control Register

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

The DA and FR bits of the Status Register are gated onto the IREQ# line by the Interrupt Enable bits for the Control Register: DAIE (bit 7) and FRIE (bit 6) respectively.

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 that are always set to 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. The Host SHALL NOT interrupt multiple write operations to the Data Register with 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 SHALL test the DA bit before initiating a Host-to-Card data transfer in order to avoid deadlock.

This C code fragment illustrates the Host side process:

```
if (Status_Reg & 0x80) /* go to module-to-host transfer (see below) */
Control_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];
}
Control_Reg = 0x00;
```

For Card-to-Host Transfers, the Host periodically 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. The Host SHALL NOT interrupt multiple read operations with 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.

Extended Channel Operation - This communication channel 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 entity can communicate with the headend via a link layer or modem function in the other entity. 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.

An extended channel application in the Card opens 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, MPEG table sections, and DSG PDUs across the CHI. Under normal operating conditions, this session will never be closed.

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 7.6-1 and Figure 7.6-2.

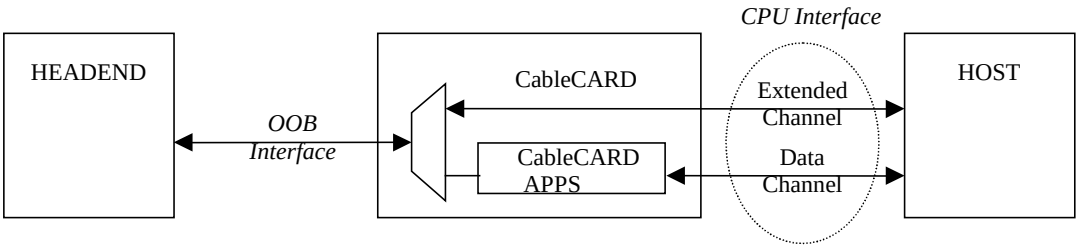


Figure 7.6-1 - Modem in-the-Card System Overview

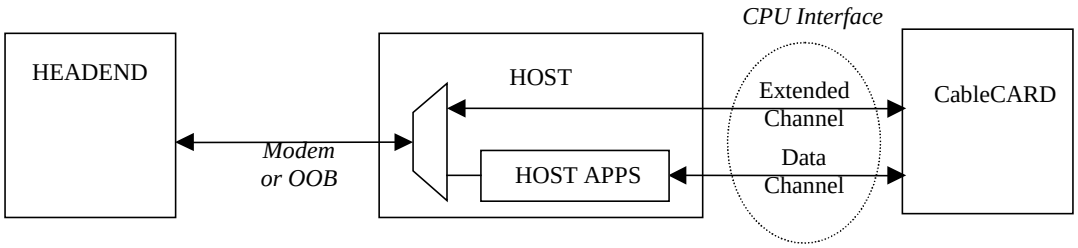


Figure 7.6-2 - Modem in-the-Host System View

The Data Channel is physically activated by CE1# (Card Enable 1), and 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 Channel 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 7.6-3 are part of the Card.

Table 7.6-4 - 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
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 buffers that may have a different size than the Data Channel buffers.

Since there are two communication channels (data channel and extended channel), the behavior of the interface is defined in such a way that when the Host sets the RS bit on either channel, the interface is reset for both channels.

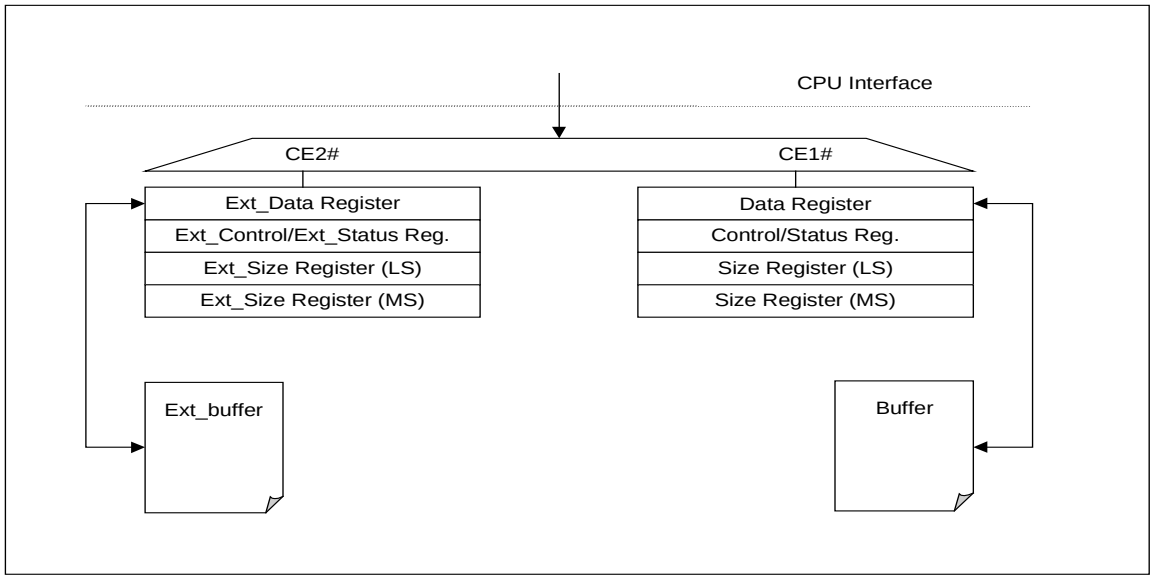


Figure 7.6-3 - Map of Hardware Interface Registers

7.6.1.1 Interrupt Operation

This section describes the interrupt operation using the DAIE and FRIE bits in the Control Register for both the Data Channel and the Extended Channel.

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#.

The following diagram shows the Card interrupt logical operation.

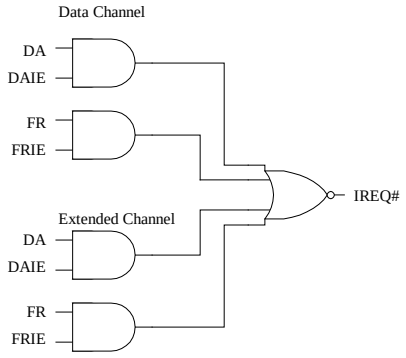


Figure 7.6-4 - CableCARD Device Interrupt Logical Operation

The Card SHALL assert IREQ# whenever the DA or FR bits are set for the data channel and the DAIE or FRIE interrupt enable bit is set.

The Card SHALL assert IREQ# whenever the DA or FR bits are set for the extended channel and the DAIE or FRIE interrupt enable bit is set.

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 will have priority over the extended channel. Additionally, the data channel will have less traffic overall than the extended channel.

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

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) is provided by the Host to signal the start of a byte of data as well as the start of a new packet. Two separate data signals are 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 length is contained in the next 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 7.6-5 - CPU Interface Packet Format

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

If the DA, HR and CR bits are set in the interface query byte, then either data channel or extended channel data will follow the packet length bytes.

7.6.2.3 Interface Flags

HR **Host Ready:** The Host SHALL set the HR (Host Ready) flag in the interface query byte when it is ready to receive only or transmit and receive data. This flag allows the Host to control the throughput of packets from the Card.

CR **CableCARD Ready:** The Card SHALL set the CR (CableCARD Ready) flag in the interface query byte when it is ready to receive only or transmit and receive data. After the RESET signal goes inactive, this signal will indicate to the Host when the Card is ready. The Card SHALL set the CR (CableCARD Ready) flag in the interface query byte 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 the EC (Extended Channel) flag in the interface query byte 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 data (from either channel) 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 will 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 will set this flag.

DA **Data Available:** Indicates to the Host/Card that data is available.

In case HR and CR is set and the Host and the Card set its DA within the same IQB, two different cases can occur.

Here are examples for each case:

1. Host indicates to send 5 data bytes, Card indicates to send 9 data bytes.

The Card will ignore the SDI signal while it sends (and the Host receives) the extra Card to Host data (data clocks 6 to 9), with the next IQB byte occurring on the clock after the 9th data byte is transferred.

2. Host indicates to send 12 data bytes, Card indicates to send 4 data bytes.

The Host will ignore the SDO signal while it sends (and the Card receives) the extra Host to Card data (data clocks 5 to 12), with the next IQB byte occurring on the clock after the 12th data byte is transferred.

ER **Error detected:** The Host or Card has detected an error in the CPU interface. If the Host detects that the Card has set the ER (Error Detected) flag in interface query byte, it SHALL perform a PCMCIA reset on the Card. The Card MAY choose to ignore the Host's ER flag in the interface query byte.

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) and the Card is ready (CR=1) and the Host is ready (HR=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.

When the Host detects that the Card has set the DA (Data Available) flag in the interface query byte and the CR flag and its HR flag is set, the Host SHALL read the following length bytes from the Card and clock all of the remaining data in.

When the Host sets its DA and HR flag and detects that the Card has set its CR flag in the interface query byte, the Host SHALL send the length bytes to the Card and send all remaining data.

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, it SHALL set the packet count following the IQB to 0.

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

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

The Host SHALL format packets as follows:

- Each packet will only be either a command or extended type, i.e., no mixing of types.
- For the command channel, only one SPDU is allowed per packet.
- For the extended channel, only one flow ID is allowed per packet.
- If a packet is not segmented, then both the F and L bits will be set.
- Packets smaller than 4,096 bytes are not to be segmented.
- If a packet is larger than 4,096 bytes, then it will be segmented into contiguous packets with the F bit set for the first packet and the L bit set for the last packet.

The Card SHALL format packets as follows:

- Each packet will only be either a command or extended type, i.e., no mixing of types.
- For the command channel, only one SPDU is allowed per packet.
- For the extended channel, only one flow ID is allowed per packet.
- If a packet is not segmented, then both the F and L bits will be set.
- Packets smaller than 4,096 bytes are not to be segmented.
- If a packet is larger than 4,096 bytes, then it will be segmented into contiguous packets, with the F bit set for the first packet, and the L bit set for the last packet.

The Card SHALL set the ER bit upon detection of an error condition (e.g. under-run). This should be considered to be a catastrophic failure during normal operation, causing the Host to perform a PCMCIA reset on the Card. There is an ER bit for the Host to set; however, the Card may 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.

All devices 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.

The Host and Card both initialize to the PCMCIA interface upon a power-up or a PCMCIA Reset condition and will, at a particular point in the sequence, change to the CableCARD device interface. This point at which the interface changes from the PCMCIA interface to the CableCARD Device interface is defined as the CableCARD Device Personality Change.

There are two possible resets that can occur in the Card interface: a hard reset (PCMCIA reset) and a soft reset (Card reset).

7.6.3.1 PCMCIA Reset

A PCMCIA reset is defined to be one in which the Host brings the RESET signal to the Card active. This causes the Card to revert to the PCMCIA interface [PCMCIA2] and will no longer route 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.

7.6.3.2 Card Reset

Card Reset is defined to occur when the Host sets the RS bit in the Card Control Register any time after the CableCARD personality change has completed. Card Reset operation is accomplished in the following manner:

To initiate a Card Reset, the Host SHALL set the RS bit in both the data channel and extended channel Control Registers.

After a Card Reset, the Card interface operation SHALL revert to the state of operation that occurs just after completion of the CableCARD Device personality change.

The Card SHALL continue to route MPEG data streams during a Card Reset operation.

The Host SHALL continue to route MPEG data streams to the Card during a Card Reset operation.

Upon initiation of Card Reset, the Card SHALL cease CA-descrambling of any MPEG data stream until a new session is opened to the CA Resource.

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

7.6.3.3 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 the Card needs to request an interface initialization, it SHALL set the IIR bit in both the Data and Extended Channel Control Registers. Upon recognition of the IIR flag being set, the Host SHALL implement a Card Reset by setting the RS bit in the control registers. The Card SHALL clear the IIR flag when the RS bit is set. The Card SHALL NOT send LPDUs to the Host after setting the IIR bit until the completion of Card Reset operation.

7.6.3.4 Detailed CableCARD Reset Operation

The following flowchart (Card Reset Sequence) is the required implementation of the Card RS operation.

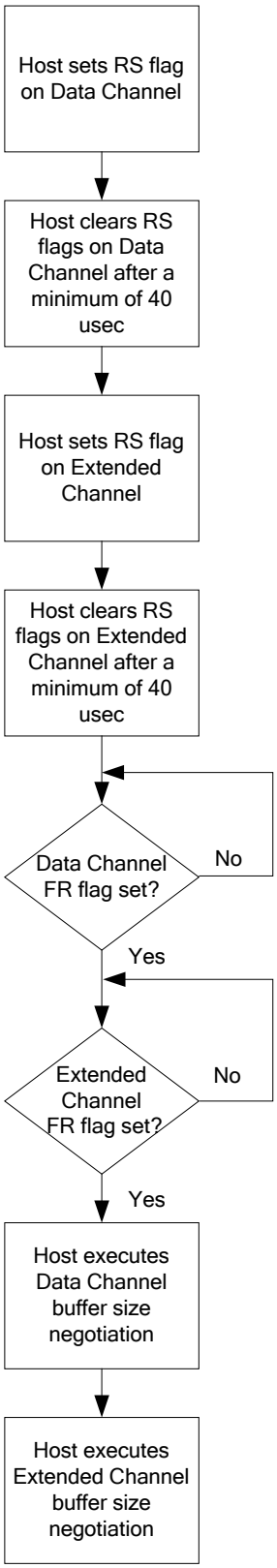


Figure 7.6-5 - Card RS Operation

7.6.3.5 Configuration Option Register

The Host SHALL support configuration of the Configuration Option Register in the Card as defined in section 4.14 of [PCMCIA2]. Host support for registers other than the Configuration Option Register is optional.

The Card SHALL support the Configuration Option Register as defined in section 4.13 of [PCMCIA2].

The Configuration Option Register (COR) in the Card is only accessible prior to the CableCARD personality change (Section 7.3.4). After the CableCARD personality change, the COR is no longer available. Any relevant configuration data is 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_INDX, from the CISTPL_CFTABLE_ENTRY tuple, the Host configures the Card into the CableCARD device mode, thus causing the CableCARD personality change.

7.6.3.6 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 will 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 will 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 will go through its initialization and then should perform a PCMCIA reset on the Card, if it can detect that the Card has not been reset. After the Card has performed its internal initialization, then the interface initialization will begin.
4. The Host has lost communications with the Card.

7.6.3.7 OOB Connection and Disconnection Behavior

The Host will not activate or transmit from its OOB transmitter until a Card is connected to the Host and initialized to the CableCARD Device interface.

The Host SHALL disable the RDC OOB transmitter when a Card is disconnected after the Host and Card have been initialized.

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

7.6.3.8 Card 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 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). Upon power-up, 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. After power-up, while the Card's READY signal pin is inactive, the Card SHALL only operate as an un-configured PCMCIA module.

   PCMCIA Reset: The RESET signal goes active for a minimum of 10 usec. Upon PCMCIA reset, 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. After

PCMCIA reset, while the Card's READY signal pin is inactive, 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. The Host 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 7.3.4 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 [NRSSB].
7. The Host SHALL write into the Configuration Option Register (COR) the value read in TPCE_INDX. Following the write cycle of the COR during a personality change, the Host SHALL switch the address signals A4-A8 to the OOB interface signals and the inband transport stream signals. The Host implements a pull-down resistor on the ETX signal to prevent spurious operation of the transmitter. It also implements 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 minimum of 10 usec after the COR write signal during a personality change, 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.

Figure 7.6-6 illustrates this operation.

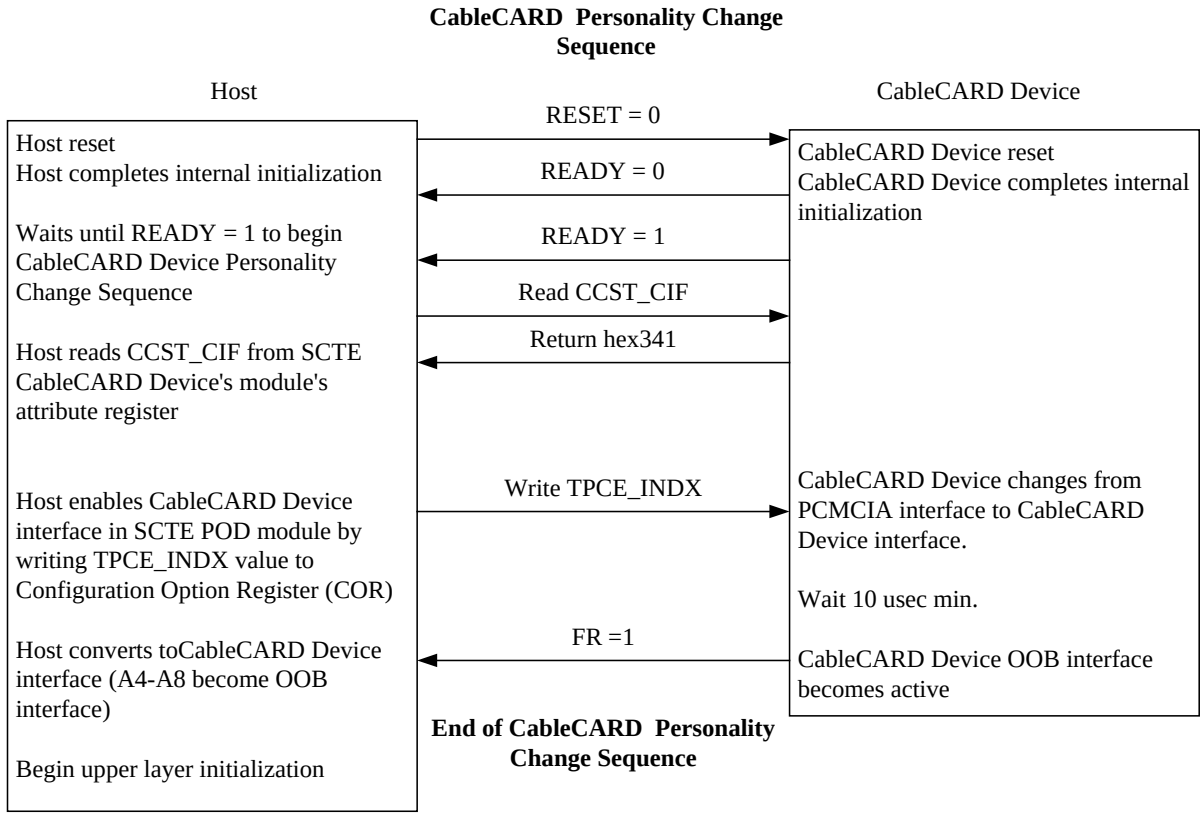


Figure 7.6-6 - CableCARD Personality Change Sequence

7.6.3.9 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 Host SHALL initialize the Data Channel first and the Extended Channel second.

7.6.3.9.1 Data Channel Initialization

The data channel SHALL be initialized by the Host as follows:

- The Host writes a '1' to the RS bit in the data channel Control Register.
- After a minimum of 40 usec, the Host writes a '0' to the RS bit in the data channel Control Register.
- The Card clears any data in the data channel data buffer and configures the interface so it can perform the data channel buffer size negotiation protocol.
- When the Reset operation is complete, the Card sets the data channel FR bit to '1'.
- The Host waits a minimum of five seconds for the Card to set the FR bit to '1' before performing a PCMCIA reset.

7.6.3.9.2 Extended Channel Initialization

The extended channel SHALL be initialized by the Host as follows:

- The Host writes a ‘1’ to the RS bit in the extended channel Control Register.
- After a minimum of 40 usec, the Host writes a ‘0’ to the RS bit in the extended channel Control/Status Register.
- The Card clears any data in the extended channel data buffer and configures the interface so it can perform the extended data channel buffer size negotiation protocol.
- When the Reset operation is complete, the Card sets the extended channel FR bit to ‘1’.
- The Host waits a minimum of five seconds for the Card to set the FR bit to ‘1’ before performing a PCMCIA reset.

7.6.3.9.3 Data Channel Buffer Size Negotiation

After initialization of the Data Channel, the Host SHALL determine the Data Channel buffer size of the Card by using 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 Data Channel 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 Data Channel 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.

The Host SHALL negotiate to the lower of its buffer size and the Card buffer size, and report that back to the Card.

7.6.3.9.4 Extended Channel Buffer Size Negotiation

The Extended Channel buffer size negotiation is the same as the data channel defined in Section 7.6.3.9.3. Note that the buffer sizes of the data and extended channels do not have to be the same. The Card SHALL support a minimum Extended Channel buffer size of 16 bytes and a maximum of 65,535 bytes. The Host SHALL support a minimum Extended Channel buffer size of 256 bytes and a maximum of 65,535 bytes.

Using the Buffer Size Negotiation protocol called out in Section 7.6.3.9.3, 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.

7.6.4 M-CARD Initialization and Operation

For the M-CARD, the following describes the operation of the CPU interface at initialization through opening the resource manager session. In this description, it is assumed that the M-CARD and Host contain 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.

In M-Mode, the CableCARD Interface SHALL be initialized as follows:

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

Table 7.6-6 - 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

- The M-CARD brings its CR flag active within 5 seconds of RESET becoming inactive.

Table 7.6-7 - 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

- 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.
- 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 second and third bytes of data from the M-CARD.
- The Host then clocks in 6 data bytes from the M-CARD.

Table 7.6-8 - 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

- The Host reads the value in its command channel input buffer, 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.
- The Host sets its DA flag and HR flag then clears its EC flag on the next Host query. The 2nd and 3rd bytes contain the length of the data, which will be 0x0009 for the open_session_response SPDU.
- The Host clocks out all data in its command channel output buffer. The M-CARD will clock this data into its command channel receive buffer.

Table 7.6-9 - 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

- The M-CARD then clears its ready flag, CR, while it processes the data in its buffer.
- Once the resource manager session is open, the M-CARD will load the ***profile_inq()*** APDU into its command channel output buffer and then bring the CR and DA flags active.

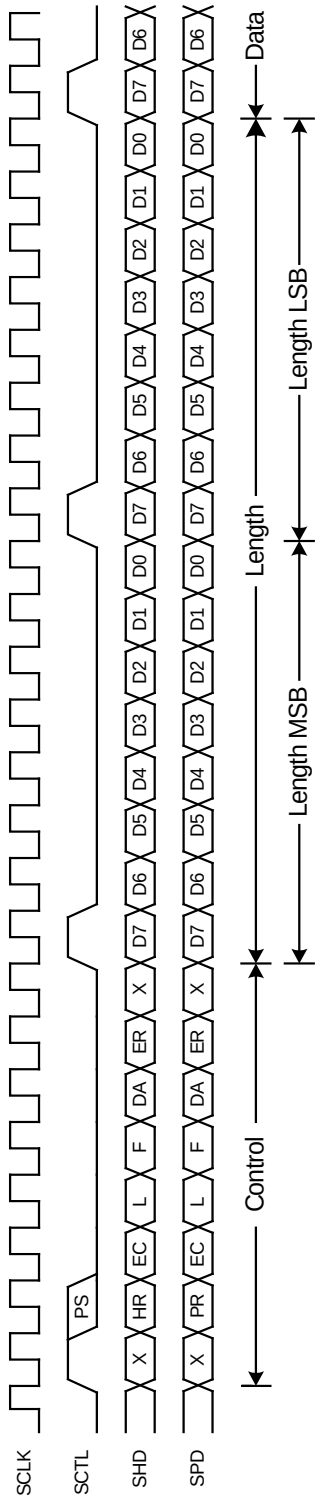


Figure 7.6-7 - M-Mode Serial Interface Protocol Diagram

7.7 Link Layer Connection

The Link Layer on the Command Interface fragments Transport Protocol Data Units (TPDU), if necessary, for sending over the limited buffer size of the Physical Layer, and reassembles received fragments. 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 are set to zero. This is illustrated in Figure 7.7-1.

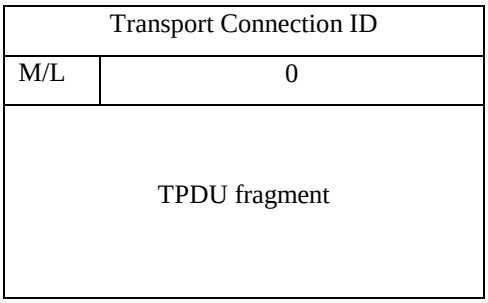


Figure 7.7-1 - Layout of Link Protocol Data Unit

Each TPDU starts 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.

No explicit initialization of the Link Layer is required.

7.8 Transport Layer Connection

The transport layer (TPDU) connection is covered in the following sections. The maximum length of the transport data is limited to 65,534 bytes.

The communication of data across the Data Channel is defined in terms of objects plus the interrupt mode extension. Card applications use the Data Channel 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 7.8-1 - Length field used by all PDUs at Transport, Session and Application Layers

Syntax	No. of bits	Mnemonic
Length_field(){		
size_indicator	1	bslbf
if (size_indicator ==0)		
length_value	7	uimsbf
else if size_indicator==1){		
length_field_size	7	uimsbf
for (i=0;i<length_field_size; i++){		
Length_value_byte	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 7.8-1 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].

7.8.1 Transport Layer

The Transport Layer of the Data 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 Card using 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 11 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 Card 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.

7.8.2 Transport protocol objects

Note: the transport layer and protocol was originally defined for a Host that supports multiple modules, or Cards. However, the OpenCable Host platform is currently specified to only support one Card at a time, so there will only be one transport connection open at a time. Many of these transport protocol objects are not used in the OpenCable Host, but are left here for possible future expansion.

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 Card 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 Card has data available to send.
9. T_RCV is sent by the Host to request that data the Card 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.

7.8.3 Transport protocol

When the Host wishes to set up a transport connection to the 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 will infer that the existing transport connection is terminated, 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, possibly preceded by a Request_T_C or Delete_T_C object if necessary.

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.

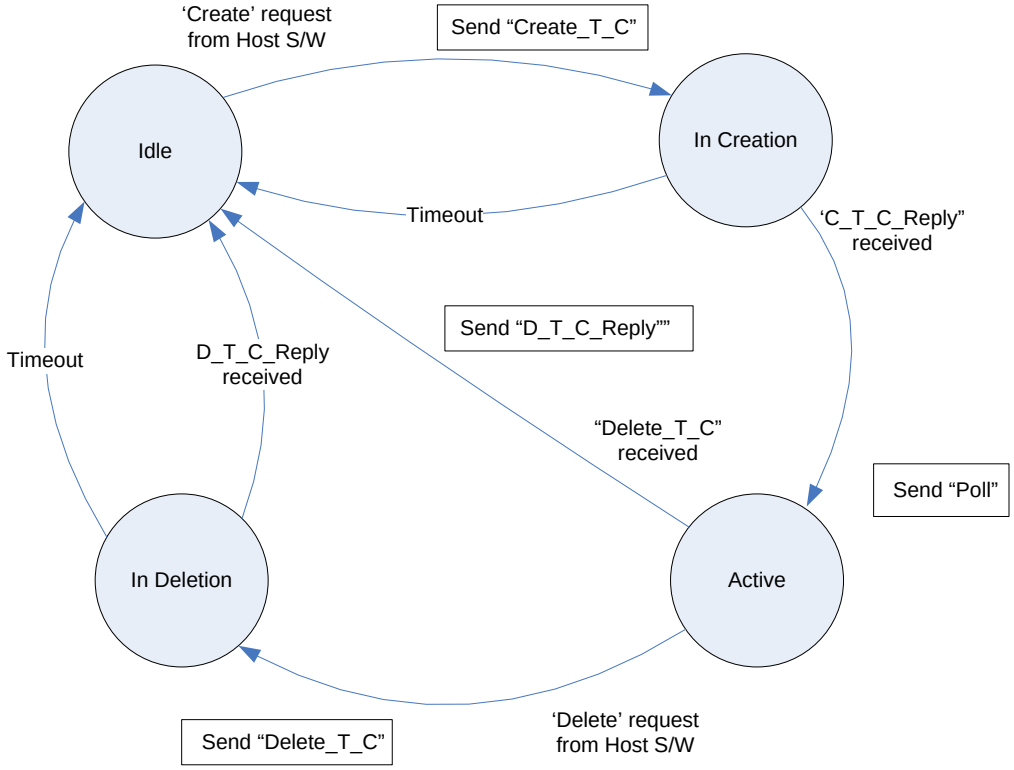


Figure 7.8-1 - State Transition Diagram - Host Side of the Transport Protocol

Table 7.8-2 - 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)

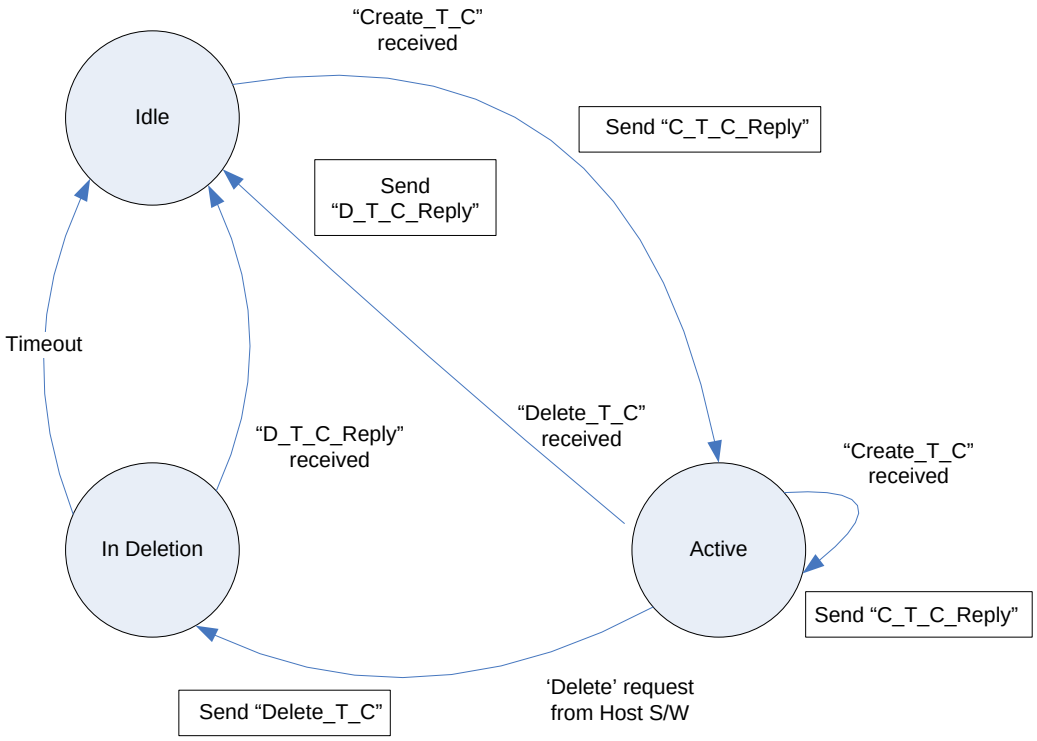


Figure 7.8-2 - State Transition Diagram - Card Side of the Transport Protocol

Table 7.8-3 - 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

An example of an object transfer sequence to create and use a Transport Connection is illustrated in Figure 7.8-3.

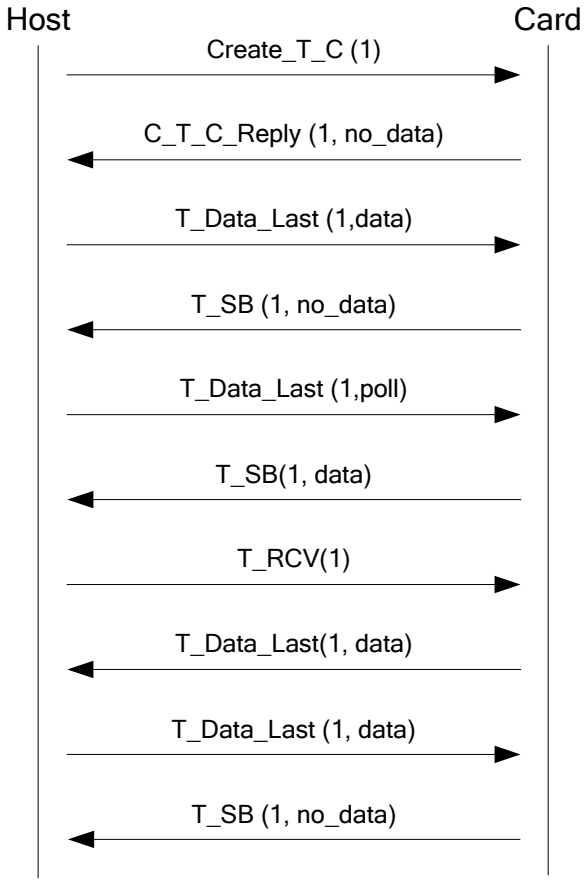


Figure 7.8-3 - Object Transfer Sequence - Transport Protocol

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 T_SB, 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.

7.8.4 Transport Protocol Objects

7.8.4.1 Command TPDU

The Host SHALL use the Command TPDU (C_TPDU) as defined in Table 7.8-4 to convey Transport Protocol objects from to the Card.

Header			Body
C_tpdu_tag	length_field	t_c_id	[data field]

Figure 7.8-4 - C_TPDU Structure

Table 7.8-4 - 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:

- 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.
- A conditional body of variable length equal to the length coded by length_field minus one.

7.8.4.2 Response TPDU

The Card SHALL use the Response TPDU (R_TPDU) as defined in Table 7.8-5 to convey Transport Protocol objects to the Host.

Header			Body		Status	
r_tpdu_tag	length_field	t_c_id	[data field]	SB_tag	length_field = 2 SB_value	t_c_id

Figure 7.8-5 - R_TPDU Structure

Table 7.8-5 - 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:

- 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.
- A conditional body of variable length equal to the length coded by length_field minus one.

- 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 Table 7.8-6 and Table 7.8-7.

Table 7.8-6 - SB_value

Bit	7	6	5	4	3	2	1	0
	DA	reserved						

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 7.8.4.10). The coding of DA indicator is given in Table 7.8-7. The seven bits in the 'reserved' field are set to zero.

Table 7.8-7 - Coding of bit 7 of SB_value

Bit 7	Meaning
0	No Data available
1	Data available

7.8.4.3 Create Transport Connection (Create_T_C)

The Host SHALL open exactly one transport connection to the Card using the Create_T_C TPDU as specified in Table 7.8-8.

Table 7.8-8 - Create Transport Connection (Create_T_C)

Syntax	Value (hex)	No. 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.

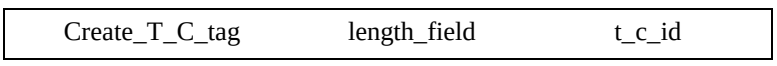


Figure 7.8-6 - Create_T_C Structure

7.8.4.4 Create Transport Connection Reply (C_T_C_Reply)

The Card SHALL respond to the Create_T_C TPDU with the C_T_C_Reply TPDU as specified in Table 7.8-9.

Table 7.8-9 - Create Transport Connection Reply (C_T_C_Reply)

Syntax	Value (hex)	No. 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.

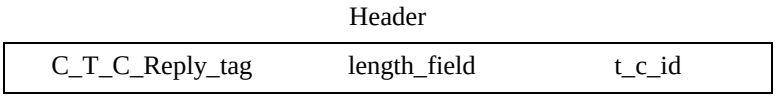


Figure 7.8-7 - C_T_C_Reply Structure

7.8.4.5 Delete Transport Connection (Delete_T_C)

Delete_T_C is issued by either the Card or the Host to delete an existing Transport Connection.

Table 7.8-10 - Delete Transport Connection (Delete_T_C)

Syntax	Value (hex)	No. of bits	Mnemonic
delete_T_C (){			
Delete_T_C_ _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.

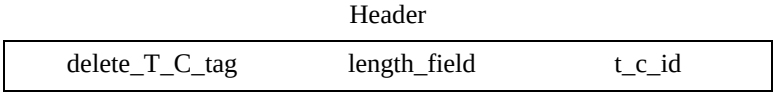


Figure 7.8-8 - Delete_T_C Structure

7.8.4.6 Delete Transport Connection Reply (D_T_C_Reply)

D_T_C_Reply is sent by either the Card or the Host in reply to Delete_T_C.

Table 7.8-11 - Delete Transport Connection Reply (D_T_C Reply)

Syntax	Value (hex)	No. 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.

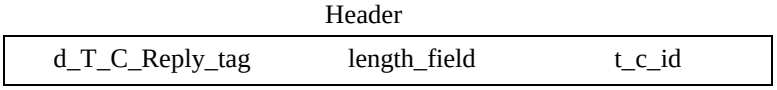


Figure 7.8-9 - D_T_C_Reply Structure

7.8.4.7 Request Transport Connection (Request_T_C)

Request_T_C is sent by the Card to the Host on an existing Transport Connection to create a new Transport Connection.

Table 7.8-12 - Request Transport Connection (Request_T_C)

Syntax	Value (hex)	No. 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.

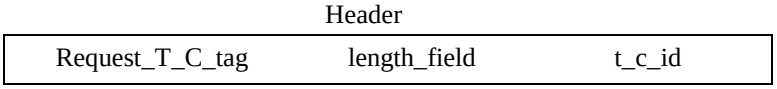


Figure 7.8-10 - Request_T_C Structure

7.8.4.8 New Transport Connection (New_T_C)

New_T_C is sent in response to Request_T_C on the same Transport Connection as the Request_T_C object and carries the transport connection identifier of the new connection.

Table 7.8-13 - New Transport Connection (New_T_C)

Syntax	Value (hex)	No. 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:

- 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.
- A mandatory body consisting of the transport connection identifier for the new connection to be established.

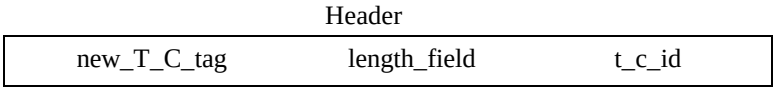


Figure 7.8-11 - Request_T_C Structure

7.8.4.9 Transport Connection Error (T_C_Error)

T_C_Error is sent in response to Request_T_C to signal that no more Transport Connections are available.

Table 7.8-14 - Transport Connection Error (T_C_Error)

Syntax	Value (hex)	No. 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:

- 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.
- A mandatory body consisting of the error code for the particular error being signaled.

Transport Connection Error Codes are defined in Table 7.8-15.

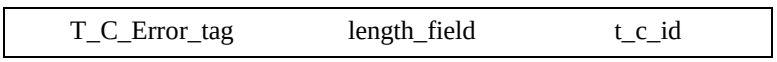


Figure 7.8-12 - T_C_Error Structure

Table 7.8-15 - Error Code Values

Error Code	Meaning
1	No transport connections available

7.8.4.10 Sending and receiving data using C_TPDU and R_TPDU

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 7.8-13.

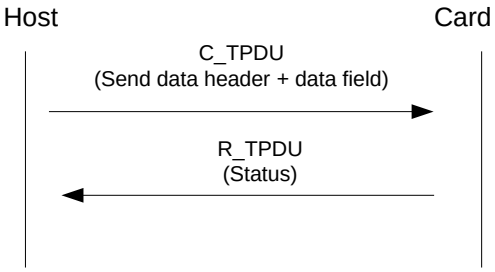


Figure 7.8-13 - Send Data command/ Response Pair

Table 7.8-16 - Send Data C_TPDU

c_TPDU_tag	^{T}data_more ^{T}data_last
length_field	Length of data field according to [ISO8825]
t_c_id	Transport connection identifier
data_field	Subset of (TLV TLV … TLV

Table 7.8-17 - Send Data R_TPDU

Status	according to Table 7.8-6

A C_TPDU with tag value Tdata_last and length_field=1 (no data field) can be issued by the Host to retrieve information given by the status byte (see polling function, Section 7.8.4.10.1).

The Receive Data C_TPDU is used by the Host to receive data from the Card. See Figure 7.8-14.

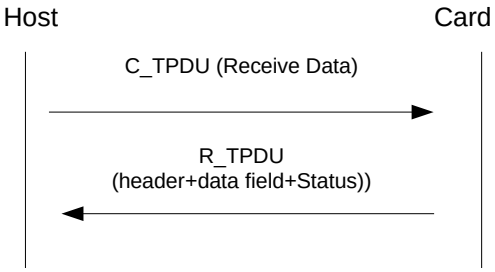


Figure 7.8-14 - Receive Data command/ Response Pair

Table 7.8-18 - Receive Data C_TPDU

c_TPDU_tag	T_{RCV}
length_field	c_TPDU_length is set to '1'
t_c_id	Transport connection identifier

Table 7.8-19 - Receive Data R_TPDU

r_TPDU_tag	Tdata_more Tdata_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 Table 7.8-6

A list of the TPDU_tag values is given in Table 7.8-20.

Table 7.8-20 - Transport Tag Values

tpdu_tag	Tag Value (hex)	Primitive or Constructed	Direction Host ↔ Card
TSB	'80'	P	←
TRCV	'81'	P	→
Tcreate_t_c	'82'	P	→
Tc_t_c_reply	'83'	P	←
Tdelete_t_c	'84'	P	↔
Td_t_c_reply	'85'	P	↔
Trequest_t_c	'86'	P	←
Tnew_t_c	'87'	P	→
Tt_c_error	'88'	P	→
Tdata_last	'A0'	C	↔
Tdata_more	'A1'	C	↔

7.8.4.10.1 Data Channel Polling Function

The transport layer polling function is used by the Host to determine if the Card has data available to be sent on the Data Channel. There is no corresponding polling function for the Extended Channel, since the transport layer is absent.

The Host SHALL provide the Data Channel polling function by periodically sending the ***C_TPDU, T_data_last()*** with length_field = 1 (t_c_id field only, no data field) to retrieve the Data Channel status byte.

The maximum period between Host executions of the Data Channel polling function SHALL be equal to 100ms.

The Card SHALL respond to the Data Channel polling function ***C_TPDU, T_data_last()*** with length_field = 1, with the ***R_TPDU, T_SB()***.

If the Card receives the Data Channel polling function C_TPDU and does not have data to send, it SHALL reply with the ***R_TPDU, T_SB()*** with SB_value field having the DA (Data Available) bit set to 0.

If the Card receives the Data Channel polling function C_TPDU and has data to send, it SHALL reply with the ***R_TPDU, T_SB()*** with SB_value field having the DA (Data Available) bit set to 1.

If the Card indicates that data is available in the ***R_TPDU, T_SB()***, the Host can issue one or more ***C_TPDUs, T_RCV()*** to retrieve the data until the DA (Data Available) bit of the SB_value field in the R_TPDU is set to 0.

The Host SHALL suspend the Data Channel polling function during data retrieval until the Card indicates that no more data is available by returning an R_TPDU with SB_value field having the DA (Data Available) bit set to 0. The Host SHALL restart the Data Channel polling function when the Card indicates that no more data is available by returning an R_TPDU with SB_value field having the DA (Data Available) bit set to 0.

The Card SHALL respond to the Data Channel polling function regardless of the status of the Card Extended Channel buffers (i.e., the Card responds with the ***R_TPDU, T_SB()***, even if the Extended Channel status register indicates that data is available for transmission).

When the Host sends the Data Channel polling function C_TPDU, it SHALL start a time-out period of five seconds, which is reset when the poll response, R_TPDU, T_SB, is received. After the Host has sent the Data Channel polling function C_TPDU and no poll response has been received within five seconds from the Card, the Host SHALL perform a PCMCIA reset. The Host SHALL NOT send any additional Data Channel polling function C_TPDUs while waiting for a poll response from the Card, even if the normal poll interval is exceeded.

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 when the Card is required to remove conditional access scrambling from the received MPEG transport stream(s). 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 [CCCP].

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 uses the Link and Transport layers as defined in Sections 5 and 7 of this document.

For 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 the mechanism by which applications on the Card communicate with and make use of resources on the Host. Resources are a mechanism for encapsulating functionality at the Application Layer and vary in the number of simultaneous sessions they can support. Some resources support only one session at a time. 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 session 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.

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

9.1.1 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.

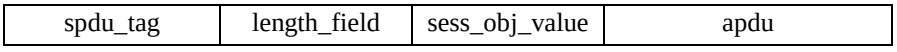


Figure 9.1-1 - SPDU Structure

The SPDU is made of two parts:

- 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.
- A conditional body of variable length that contains an 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.

Only the session_number SPDU is followed by a data_byte field 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.

9.1.2 Session Layer Protocol

The session objects are described in general terms here.

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).

The dialogue in the session layer is initiated by the Card. One example is illustrated in Figure 9.1-2, where a Card requests to open a resource provided by the host.

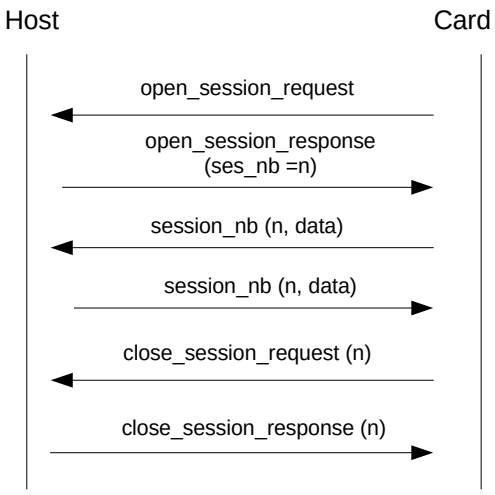


Figure 9.1-2 - Object Transfer Sequence - Transport Protocol

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.

9.1.2.1 Open Session Request (open_session_request)

The ***open_session_request()*** SPDU is issued by the Card in order to request the opening of a session to a specific resource provided by the Host. The resource_identifier field in the ***open_session_request()*** SPDU sent by the Card SHALL match the class and type of a resource that the Host has already declared support for in the ***profile_reply()*** APDU. Should the Card request version = 0 in the ***open_session_request()*** SPDU, the Host SHALL use the resource version number it reported in the ***profile_reply()*** APDU.

Once the Card has obtained the types and versions of all resources supported by the Host, it must determine the highest version of each resource common to both devices.

If the highest version of a resource that the Card supports is equal to the version reported by the Host in the ***profile_reply()*** APDU, the Card SHALL request this resource version in the ***open_session_request()*** SPDU.

If the highest version of a resource that the Card supports is greater than the version reported by the Host in the ***profile_reply()*** APDU, the Card SHALL request the resource version reported by the Host, in the ***open_session_request()*** SPDU.

If the highest version of a resource that the Card supports is less than the version reported by the Host in the ***profile_reply()*** APDU, the Card SHALL request the highest resource version it supports, in the ***open_session_request()*** SPDU.

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.

9.1.2.2 Open Session Response (open_session_response)

The Host SHALL issue an ***open_session_response()*** SPDU to the Card in response to an ***open-session_request()*** from the Card, with the syntax as defined in Table 9.1-3, 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`	`8`	`uimsbf`
`  length_field() /* always equal to 0x07 */`		
`  session_status`	`8`	`uimsbf`
`  resource_identifier()`		
`  session_nb`	`16`	`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 resource identifier of the version requested by the Card

session_nb A unique 16-bit integer number allocated by the Host for the requested session. When the session could not be opened (session_status ≠ 0x00), this value has no meaning. Note that the value 0x00 is reserved.

The Host SHALL NOT use a value of 0x00 for the session_nb (session number) in the ***open_session_response()*** SPDU.

The Host SHALL allocate a unique session_nb for each new session requested by the Card within resources and across all resources.

The Card SHALL use the session_nb allocated in the ***open_session_response()*** SPDU for all subsequent exchanges of APDUs between the Card and the Host until the session is closed.

The Host SHALL use the session_nb allocated in the ***open_session_response()*** SPDU for all subsequent exchanges of APDUs between the Card and the Host until the session is closed.

9.1.2.3 Close Session Request (close_session_request)

Either the Host or the Card can issue a ***close_session_request()*** to close a session.

The Host SHALL issue a ***close_session_request()*** SPDU with the syntax as defined in Table 9.1-4 to close a session.

The Card SHALL issue a ***close_session_request()*** SPDU with the syntax as defined in Table 9.1-4 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.

9.1.2.4 Close Session Response (close_session_response)

Either the Host or the Card will issue a ***close_session_response()*** in response to a ***close_session_request()***.

The Host SHALL issue the ***close_session_response()*** SPDU after receiving a ***close_session_request()*** SPDU.

The 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
  length_field() /* always equal to 0x03 */
  session_status
  session_nb
}
	
	8
	
	8
	16
		
		uimsbf
		
		uimsbf
		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.

9.1.2.5 Session Number (session_number)

The Host SHALL precede an APDU with a ***session_number()*** SPDU with the syntax as defined in Table 9.1-6.

The Card SHALL precede an APDU with a ***session_number()*** SPDU with the syntax as defined in Table 9.1-6.

Table 9.1-6 - session_number() Syntax

Syntax	No. of Bits	Mnemonic

session_number() {
  session_number_tag
  length_field() /* always equal to 0x02 */
  session_nb
}
	
	8
	
	16
		
		uimsbf
		
		uimsbf

session_number_tag 0x90

session_nb The 16-bit integer value assigned to the session.

9.1.2.6 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	→
close_session_request	0x95	↔
close_session_response	0x96	↔
session_number	0x90	↔

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.

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 the private_resource_definer, defined to be the CHICA-assigned manufacturer number (see [CCCP]). 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																															
31							24	23							16	15								6	5						0
type	resource_class															resource_type									resource_version						

Table 9.2-2 - Private Resource Identifier

Bit																															
31							24	23							16	15							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	2	uimsbf
if (resource_id_type != 0x3) {		
resource_class	14	uimsbf
resource_type	10	uimsbf
resource_version	6	uimsbf
}		
else {		
private_resource_definer	10	uimsbf
private_resource_identity	20	uimsbf
}		
}		

Applications on the Card make use of resources on the Host to perform tasks for the user of the Host.

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 as shown in Figure 9.3-1.

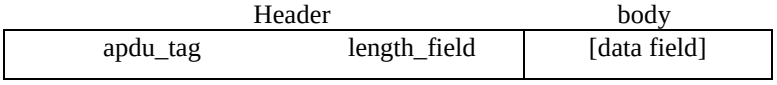


Figure 9.3-1 - APDU Structure

The Host SHALL send only a single APDU within the body of a single SPDU.

The Card SHALL send only a single APDU within the body of a single SPDU.

Table 9.3-1 - APDU Structure Syntax

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

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

Table 9.3-2 - Resource Identifier Values

Resource (Type = 0 unless noted)	Resource Class	Resource Type	Resource Version	Resource identifier
Resource Manager	1	1	1	0x00010041
Application Information	2	2	1	0x00020081
Application Information	2	2	2	0x00020082
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
MMI	64	2	1	0x00400081
Low Speed Communication[2]	96	321	3	0x00605043
Low Speed Communication[2]	96	513	3	0x00608043
Homing[1]	17	1	2	0x00110042
Copy Protection	176	3	1	0x00B000C1
Copy Protection*	See [CCCP] for the current Resource Class, Type and Version for this resource.			
Specific Application Support	144	1	2	0x00900042
Generic Feature Control	42	1	1	0x002A0041
Generic Feature Control	42	1	2	0x002A0042
Generic Feature Control	42	1	3	0x002A0043
Generic Feature Control	42	1	4	0x002A0044
Headend Communication	44	1	1	0x002C0041
Extended Channel	160	1	1	0x00A00041
Extended Channel	160	1	2	0x00A00042
Extended Channel	160	1	3	0x00A00043
Extended Channel	160	1	4	0x00A00044
Extended Channel	160	1	5	0x00A00045
Extended Channel	160	1	6	0x00A00046
Generic IPPV Support	128	2	1	0x00800081
Generic Diagnostic Support	260	1	2	0x01040042
Generic Diagnostic Support*	260	2	1	0x01040081
Generic Diagnostic Support*	260	2	2	0x01040082
System Control	43	1	1	0x002B0041
System Control***	43	1	2	0x002B0042
System Control***	43	1	3	0x002B0043
System Control	43	2	1	0x002B0081
CARD RES*	38	3	1	0x002600C1
DSG****	4	1	1	0x00040041
Host Addressable Properties	256	1	1	0x01000041
Card MIB Access	90	1	1	0x005A0041
Reserved**	177	1	1	0x00B10041
Reserved*****	Type 1, 0-255	N/A	N/A	N/A

Notes:
*For the Card operating in M-Mode.
**Reserved.
***These versions have been deprecated.
****This mode will only be implemented on M-Cards.
*****Reserved for use by ATIS. NOTE: Type = 1.
1. The Homing resource is defined in 9.18.4.
2. The Resource identifier delivered by a Host will 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 is 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.

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

apdu_tag	Tag value	Resource	Direction Host ↔ Card
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	←

apdu_tag	Tag value	Resource	Direction Host ↔ Card	
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.	→	
			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
Reserved	0x9F8F00 - 0x9F8F07	Generic IPPV Support		
inquire_DSG_mode	0x9F9100	DSG	→	→
set_DSG_mode	0x9F9101	DSG	←	←
DSG_error	0x9F9102	DSG	←	N/A
DSG_message	0x9F9103	DSG	→	N/A
DSG_directory	0x9F9104	DSG	←	N/A
send_DCD_info	0x9F9105	DSG	→	N/A
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	→	

apdu_tag	Tag value	Resource	Direction Host ↔ Card
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_server_query	0x9F9A05	Specific Application Support	↔
SAS_server_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	→
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	→

apdu_tag	Tag value	Resource	Direction Host ↔ Card
code_version_table2	0x9F9C02	System Control	←
code_version_table_reply	0x9F9C03	System Control	→
host_download_control	0x9F9C04	System Control	→
code_version_table	0x9F9C05	System Control	←
diagnostic_req	0x9FDF00	Generic Diagnostic Support	←
diagnostic_cnf	0x9FDF01	Generic Diagnostic Support	→
host_reset_vector	0x9F9E00	Headend Communication	←
host_reset_vector_ack	0x9F9E01	Headend Communication	→
host_properties_req	0x9F9F01	Host Addressable Properties	→
host_properties_reply	0x9F9F02	Host Addressable Properties	←
snmp_req	0x9FA000	Card MIB Access	→
snmp_reply	0x9FA001	Card MIB Access	←
get_rootOid_req	0x9FA002	Card MIB Access	→
get_rootOid_reply	0x9FA003	Card MIB Access	←
Reserved***	0x9FE000-0x9FEF7F	N/A	N/A

Notes:
* Messages defined in EIA 679-B Part B [NRSSB].
** Reserved.
*** Reserved for use by ATIS.

9.3.1 Interface Resource Loading

Table 9.3-4 - Host-Card Interface Resource Loading

Item	Name	Maximum sessions at one time	Closes
1	Resource Manager	1	No
2	MMI	1	No
3	Application Info	1	No
4	Low Speed Communication	1	Yes
5	Conditional Access Support	1	No
6	Copy Protection	1	No
7	Host Control	1	No
8	Extended Channel Support	1	No
9	DSG	1	No
10	Specific Application Support	32	Yes
11	Generic Feature Control	1	No
12	Homing	1	Yes
13	Generic Diagnostic Support	1	No
14	System Time	1	Yes
15	System Control	1	No
16	CARD RES (Card Resource)	1	No
17	Authorized Service Domain	1	No
18	Headend Communication	1	No
19	Host Addressable Properties	1	No
20	Card MIB Access	1	No

NOTES:

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. Transport Connection is never to be closed. The Host SHALL open only one Transport Connection session at a time.

9.4 Resource Manager

The Resource Manager is a resource provided by the Host. There is only one type in the class. The Host SHALL provide a Resource Manager resource that supports a maximum of one Resource Manager session. The Resource Manager 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.

The first session that the Card opens SHALL be to the Resource Manager resource. After a session to the Resource Manager resource is open, the Host sends the ***profile_inq()*** APDU to the Card. In response to the ***profile_inq()*** APDU from the Host, the Card sends the ***profile_reply()*** APDU, which may contain a list of resource identifiers. In response to the ***profile_reply()*** APDU, the Host sends the ***profile_changed()*** APDU to the Card. The Card responds to the ***profile_changed()*** APDU with the ***profile_inq()*** APDU. The Host responds to the ***profile_inq()*** APDU with the ***profile_reply()*** APDU containing a list of supported resources.

In the rare event that a resource is modified on the Host, the Host SHALL reset the Card using any one of the available reset types.

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 Card SHALL send the ***profile_inq()*** APDU, as defined in Table 9.4-3, in response to the ***profile_changed()*** APDU.

The Host SHALL send the ***profile_inq()*** APDU, as defined in Table 9.4-3, after a session to the Resource Manager resource has been opened.

Table 9.4-3 - profile_inq() APDU Syntax

Syntax	No. of Bits	Mnemonic

`profile_inq() {`
`  profile_inq_tag`
`  length_field() /* always = 0x00 */`
`}`
	24	uimsbf

profile_inq_tag 0x9F8010

9.4.2 profile_reply()

The Host SHALL send the ***profile_reply()*** APDU, as defined in Table 9.4-4, in response to a ***profile_inq()*** APDU. The resource_identifier field in the ***profile_reply()*** APDU sent by the Host reports the *type(s)* and version of each resource applicable to its operating mode (S/M-Mode). The Host SHALL report the highest supported version number within each *type* in the ***profile_reply()*** APDU.

The resource_identifier field in the ***profile_reply()*** APDU sent by the Host MAY report more than one *type* of each resource applicable to its operating mode (S/M-Mode), if multiple *types* exist.

The Host SHALL provide support for ALL versions up to the version reported in the ***profile_reply()*** APDU within each *type*, for all resources applicable to its operating mode (S/M-Mode), in order to ensure backward compatibility with Cards that may not support the current implementation of resources.

The Host MAY provide support for previous, non-deprecated *types* and versions of a particular resource, applicable to its operating mode (S/M-Mode).

Note: An M-Mode-only Host does not have to implement the S-Mode version of a resource, if one exists, and an S-Mode-only Host does not have to implement the M-Mode version of a resource.

The Card SHALL send the ***profile_reply()*** APDU, as defined in Table 9.4-4, in response to a ***profile_inq()*** APDU. The resource_identifier field in the ***profile_reply()*** APDU sent by the Card MAY contain a null list of resource identifiers.

The Card SHALL provide support for all previous, non-deprecated *types*, in addition to the highest *type* defined in Table 9.3-2 and ALL previous, non-deprecated versions of a particular resource, in order to ensure backward compatibility with Hosts that may not support the latest implementation of resources. The M-Card SHALL support both the S-Mode and M-Mode versions of a resource, if one exists.

Table 9.4-4 - profile_reply() APDU Syntax

Syntax	No. of Bits	Mnemonic

`profile_reply() {`
`  profile_reply_tag`
`  length_field()`
`  for (i=0; i<N; i++) {`
`  resource_identifier()`
`  }`
`}`
	24	uimsbf

profile_reply_tag 0x9F8011

N length divided by 4

9.4.3 profile_changed()

The Host SHALL send the ***profile_changed()***, as defined in Table 9.4-5, in response to the ***profile_reply()*** APDU.

Table 9.4-5 - profile_changed() APDU Syntax

Syntax	No. of Bits	Mnemonic

`profile_changed() {`
`  profile_changed_tag`
`  length_field() /* always = 0x00 */`
`}`
	24	uimsbf

profile_changed_tag 0x9F8012

9.5 Application Information

The Host SHALL provide the Application Information Resource using identifier(s) as defined in Table 9.5-1. The Card SHALL open only one session to the Application Information Resource 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

Two Application Information resource versions are identified 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.

The Host SHALL keep the session to the Application Information Resource open during normal operation.

The Card SHALL keep the session to the Application Information Resource 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
Application Info	S-Mode/M-Mode	2	2	2	0x00020082

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 as defined in Table 9.5-3 to the Card. The Host includes its display capabilities in the ***application_info_req()*** APDU. The Card replies to an ***application_info_req()*** with an ***application_info_cnf()*** APDU to describe its supported applications.

Table 9.5-3 - application_info_req() APDU Syntax

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

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

display_type_support 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.

Data_entry_support 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

HTML_support Defines the HTML support capability of the Host.

The Host SHALL support at a minimum the Baseline HTML profile as defined in Annex A.

0x00 Baseline Profile
0x01 Custom Profile
0x02 HTML 3.2
0x03 XHTML 1.0
0x04-0xFF Reserved

link_support Defines whether the Host can support single or multiple links.

0x00 One link
0x01 Multiple links
0x02-0xFF Reserved

form_support 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

table_support 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 send the ***application_info_cnf()*** APDU as defined in Table 9.5-4 after receiving an ***application_info_req()*** from the Host. The S-Card and the M-Card operating in S-Mode SHALL use "pod" in the application_url_byte field of the ***application_info_cnf()*** APDU. A properly constructed URL would be:

pod:///apps/filename.html

The M-Card operating in M-Mode SHALL use "CableCARD" in the application_url_byte field of the ***application_info_cnf()*** APDU. A properly constructed URL would be:

CableCARD///apps/filename.html

Table 9.5-4 - application_info_cnf() APDU (Type 2, Version 1 and Version 2) Syntax

Syntax	No. of Bits	Mnemonic
application_info_cnf() {		
application_info_cnf_tag	24	uimsbf
length_field()		
Card_manufacturer_id	16	uimsbf
Card_version_number	16	uimsbf
if (resource_version >= 2) {		
Card_MAC_address	48	uimsbf
Card_serial_number_length	8	uimsbf
for (j=0; j<Card_serial_number_length; j++) {		
Card_serial_number_byte	8	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

Card_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	Cisco (Scientific-Atlanta)
0x02XX	NDS
0x03XX	Conax
0x04XX	Nagravision
0x0500-0xFFFF	Reserved for future manufacturers

Card_version_number Privately defined by the Card manufacturer.

Card_MAC_address Card MAC address in hex.

Card_serial_number_length Length in bytes for the Card serial number.

Card_serial_number_byte Vendor specific card serial number text.

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
0x08	Network Interface (DSG)
0x09	Conditional Access Network Handler (CANH)
0x0A-0xFF	Reserved for future applications

application_version_number Defined by the Card application supplier.

The Card SHALL upgrade the application_version_number in the ***application_info_cnf()*** APDU each time the Card application software is modified according to the Card Firmware Upgrade Host Interface. See Section 9.18 of this document.

application_name_length Length of the application name in the following for loop. The maximum value is 32.

The application_name_length in the ***application_info_cnf()*** APDU SHALL be equal to 0 for applications that do not have an MMI interface.

application_name_byte 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 provided in the ***application_info_cnf()*** APDU. 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 menu.

application_url_length Length of the application url in the following for loop.

application_url_byte Defines the URL of the Card application's top-level HTML page in the Card memory. The Host uses the application URL provided in the ***application_info_cnf()*** APDU in a ***server_query()*** 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.

The Host MAY display the application URL provided in the ***application_info_cnf()*** APDU in its menu.

9.5.3 server_query()

The Host SHALL send a ***server_query()*** APDU as defined in Table 9.5-5 to the Card to request the information in the Card file server system pointed to by the provided URL. The Host SHALL use the complete URL sent by the Card in the ***application_info_cnf()*** APDU in the url_byte field of the ***server_query()*** APDU. 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.

Table 9.5-5 - server_query() APDU Syntax

Syntax	No. of Bits	Mnemonic

server_query() {
  server_query_tag
  length_field()
  transaction_number
  header_length
  for (i=0; i<header_length; i++) {
  header_byte
  }
  url_length
  for (i=0; i<url_length; i++) {
  url_byte
  }
}
	
	24
	
	8
	16
	
	8
	
	16
	
	8
	
		uimsbf
		
		uimsbf
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf

server_query_tag 0x9F8022

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.

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 that does not support HTTP headers SHALL set header_length in the ***server_query()*** APDU to 0x00.

The Card MAY ignore the header_byte in the ***server_query()*** APDU.

url_length The number of URL bytes in the following for loop.

url_byte Each url_byte is an octet of a parameter that defines a protocol, domain, and location for the transfer of data.

The scheme designator in the URL (first part of the URL) for the Host communicating to an S-Card or an M-Card operating in S-Mode is “pod”. The scheme designator in the URL (first part of the URL) for the Host communicating to an M-Card operating in M-Mode 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 S-Card or M-Card operating in S-Mode, the properly constructed URL would be:

pod:///apps/user/program_guide/menu.html

In order to request the file menu.html from the directory/apps/user/program_guide on M-Card operating in M-Mode, 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 to the S-Card or M-Card operating in S-Mode would be:

pod:///path/file?name1=value1&name2=value2&...

A properly constructed URL to the M-Card operating in M-Mode 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. In response to a ***server_reply()*** APDU from the Card, the Host MAY be send data to the Card through the use of name-value pairs, preceded by "?" and separated by "&", as part of the URL in a ***server_query()*** APDU.

9.5.4 server_reply()

The Card SHALL send a ***server_reply()*** APDU as defined in Table 9.5-6 in response to a ***server_query()*** APDU from the Host.

Table 9.5-6 - server_reply() APDU Syntax

Syntax	No. of Bits	Mnemonic
server_reply() {		
server_reply_tag	24	uimsbf
length_field()		
transaction_number	8	uimsbf
file_status	8	uimsbf
header_length	16	uimsbf
for (i=0; i<header_length; i++) {		
header_byte	8	uimsbf
}		
file_length	16	uimsbf
for (i=0; i<url_length; i++) {		
file_byte	8	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 Card that does not support HTTP headers SHALL set the header_length in the ***server_reply()*** APDU to 0x00. The Host MAY ignore the header_byte field in the ***server_reply()*** APDU.

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. A Host using the FDC/RDC OOB as defined in [SCTE55-1] or [SCTE55-2] SHALL forward all data 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 Host SHALL report a Low_Speed_Communication Identifier that is 0x00605043 for a device with Cable Return Channel, as defined in Table 9.6-2.

The Host SHALL report a Low_Speed Communication Resource Identifier that is 0x00608043 for a device with an embedded DOCSIS cable modem, as defined in Table 9.6-2.

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[1]	Yes	None
Yes	Yes	No	0x00605043
Yes	Yes	Yes	0x00608043

1. This variation is not permitted

9.7 CA Support

This resource provides a set of objects to support Conditional Access applications. The Host SHALL provide a Conditional Access Support Resource using identifier(s) as defined in Table 9.7-1 with a maximum of one session. The Card SHALL open a single session of the CA Support Resource using identifier(s) defined in Table 9.7-1 after

the Application Information session initialization has completed. The Host sends a ***ca_info_inquiry()*** APDU to the CA application running on the Card, which responds by returning a ***ca_info()*** APDU 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 APDUs.

In the case of multiple transport streams, there is a local transport stream ID (LTSID) included in the APDUs. The Host SHALL keep the Conditional Access Support session open at all times during normal operation. The Card SHALL keep the Conditional Access Support session open at all times during normal operation.

Table 9.7-1 - CA 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.

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 Host SHALL send the ***ca_info_inquiry()*** APDU as defined in Table 9.7-3 to the Card after the CA session is opened.

Table 9.7-3 - ca_info_inquiry() APDU Syntax

Syntax	No. of Bits	Mnemonic

`ca_info_inquiry() {`
`  ca_info_inquiry_tag`
`  length_field() /* always = 0 */`
`}`
	24	uimsbf

ca_info_inquiry_tag 0x9F8030

9.7.2 ca_info

In response to a ***ca_info_inquiry()*** APDU, the Card SHALL send the ***ca_info()*** APDU as defined in Table 9.7-4 with length_field = 0x02 and a single CA_system_id.

Table 9.7-4 - ca_info() APDU Syntax

Syntax	No. of Bits	Mnemonic

ca_info() {
  ca_info_tag
  length_field() /* always = 0x02 */
  for (i=0; i<N ;i++) {
  CA_system_id
  }
}
	24
	
	16
		uimsbf
		
		uimsbf

ca_info_tag 0x9F8031

CA_system_id CA system id supported by the Card.

9.7.3 ca_pmt

The ***ca_pmt()*** APDU consists of entitlement control information extracted from the Program Map Table (PMT) by the Host and sent to the Card. This control information allows the 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. In the case that the selected program does not contain CA descriptors, the ca_pmt contains no descriptors. The Host SHALL NOT include any descriptors other than CA descriptors in the ***ca_pmt***() APDU.

CA descriptors may be included in the ***ca_pmt( )*** APDU at the program level and at the elementary stream level. The Card SHALL apply program level CA descriptors in the ***ca_pmt()*** APDU to all elementary streams that have no elementary stream level CA descriptor. When a CA descriptor is present in the ***ca_pmt()*** APDU at the elementary stream level, the Card 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 Card of which PID stream carries the Entitlement Control Messages associated with each program or elementary stream. In S-Mode, the Host MAY elect to pass all CA descriptors to the Card, or it may filter the CA descriptors based on the rules for M-Mode given below.

When operating in M-Mode, the Host SHALL always use the CA_system_id received in the ***ca_info()*** APDU to filter CA descriptors that are passed to the Card as follows:

The filtering process SHALL occur once at the program level and once for each elementary stream of the PMT.

To filter CA descriptors at any given level, the Host SHALL compare the CA_system_id received in the ***ca_info()*** APDU to the CA_system_id present in each CA descriptor it encounters, passing only the first matching CA descriptor to the Card.

If no CA descriptor matches at a given level, the Host SHALL NOT pass a CA descriptor for that level.

The Host SHALL present a CA descriptor in the ***ca_pmt()*** APDU at the same program and elementary stream levels as it originally appeared in the program's PMT. The Host SHALL NOT replicate program level descriptors at the elementary stream level in the ***ca_pmt()*** APDU, even if the Host elects to only request decryption of individual streams from that program.

The Host SHALL send a new ***ca_pmt()*** APDU with ca_pmt_cmd_id 0x01 (ok_descramble) or 0x03 (query), as defined in Table 9.7-5 for S-Mode, or Table 9.7-6 for M-Mode, when:

- the user selects a different program
- the PMT version_number changes
- the PMT current_next_indicator changes

The Host SHALL send the ***ca_pmt()*** APDU with no CA descriptor when none is present.

The Host SHALL send a new ***ca_pmt()*** APDU with ca_pmt_cmd _id 0x04 (not_selected) as defined in Table 9.7-5 for S-Mode, or Table 9.7-6 for M-Mode, when the Host no longer requires CA-decryption of a previously-selected program, such as in the case of tuning to a different program.

Table 9.7-5 - S-Mode ca_pmt() APDU Syntax (Resource Type 1 Version 2)

Syntax	No. of Bits	Mnemonic
ca_pmt() {		
ca_pmt_tag	24	uimsbf
length_field()		
ca_pmt_list_management	8	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
reserved	4	bslbf
program_info_length	12	uimsbf
if (program_info_length != 0) {		
ca_pmt_cmd_id	8	uimsbf
for (i=0; i<N; i++) {		
CA_descriptor() /* program level */		
}		
}		
for (i=0; i<N1; i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID /* elementary stream PID*/	13	uimsbf
reserved	4	bslbf
ES_info_length	12	uimsbf
if (ES_info_length != 0) {		
ca_pmt_cmd_id /* at ES level */	8	uimsbf
for (i=0; i<N2; 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 program_number has already been received this the action is identical to “update”.
- 0x05 update - The ca_pmt is already on the list but either the version number or the curent_next_indicator has changed.

0x06-0xFF Reserved

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].

program_info_length Number of bytes in the program_info field.

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 CableCARD device can start descrambling the program or start an MMI dialog immediately.
- 0x02 ok_mmi - The application can start a MMI dialog but will 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 must guarantee that a MMI session can be opened by the CableCARD device.
- 0x03 query - The Host expects to receive a ***ca_pmt_reply()*** APDU. The Card will 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 CableCARD device that the Host no longer requires that the CableCARD device attempt to descramble the service. The CableCARD device will 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.

Table 9.7-6 - M-Mode ca_pmt() APDU Syntax (Resource Type 2 Version 1)

Syntax	No. of Bits	Mnemonic
ca_pmt() {		
ca_pmt_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_pmt_cmd_id	8	uimsbf
reserved	4	bslbf
program_info_length	12	uimsbf
if (program_info_length != 0) {		
CA_descriptor() /* program level */		
}		
for (i=0; i<N1; i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID /* elementary stream PID*/	13	uimsbf
reserved	4	bslbf
ES_info_length	12	uimsbf
if (ES_info_length != 0) {		
ca_pmt_cmd_id /* at ES level	8	uimsbf
for (i=0; i<N2; 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 is 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 will 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 must 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 will 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 will close any MMI dialog it has opened.

0x05-0xFF Reserved

program_info_length Number of bytes in the program_info field. If non-zero, is the number of bytes in the CA_descriptor.

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.

The following requirements apply to Cards operating in both S-Mode and M-Mode.

Upon receipt of a ***ca_pmt()*** APDU with a **ca_pmt_cmd_id** of 0x02 (ok_mmi), the Card application can start an MMI dialog but SHALL NOT start descrambling before reception of a new ***ca_pmt()*** APDU with ca_pmt_cmd_id set to “ok_descrambling”.

If the Host sends a ***ca_pmt()*** APDU with a **ca_pmt_cmd_id** of 0x02 (ok_mmi), it SHALL allow an MMI session to be opened by the Card.

Upon receipt of a ***ca_pmt()*** APDU with a **ca_pmt_cmd_id** of 0x03 (query), the Card SHALL NOT start descrambling before reception of a new ***ca_pmt()*** APDU with ca_pmt_cmd_id set to “ok_descrambling”.

Upon receipt of a ***ca_pmt()*** APDU with a **ca_pmt_cmd_id** of 0x03 (query), the Card SHALL NOT start an MMI dialog before reception of a new ***ca_pmt()*** APDU with ca_pmt_cmd_id set to “ok_mmi”.

Upon receipt of a ***ca_pmt()*** APDU with a **ca_pmt_cmd_id** of 0x04 (not selected), the Card SHALL close any MMI dialog it has opened.

9.7.3.1 M-Mode Processing Rules for ca_pmt()

The program_index tracks the program number and LTSID assigned to a particular CA resource in the M-CARD. The Host provides the program_index in the ***ca_pmt()*** APDU to allow the M-CARD to maintain a similar index table to track the assignments that it receives from the Host. The Host SHALL update assignments to each program_index as old programs are replaced by new programs or decryption is no longer required, 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	Trans-action ID	ltsid	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	Trans-action ID	ltsid	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.

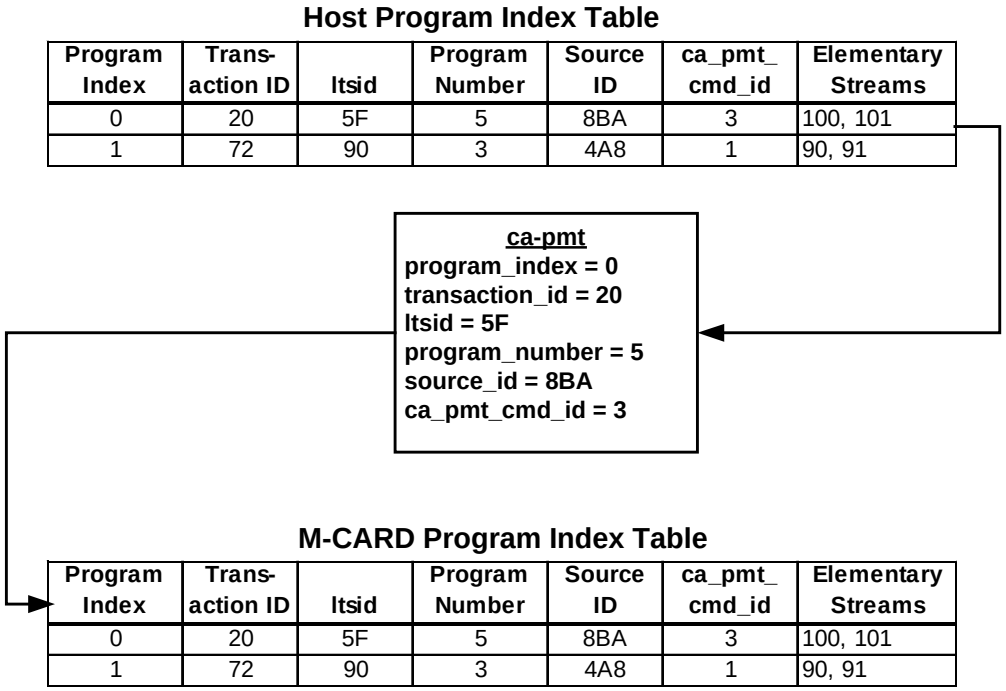


Figure 9.7-2 - Program Index Table 2

The M-CARD does 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.

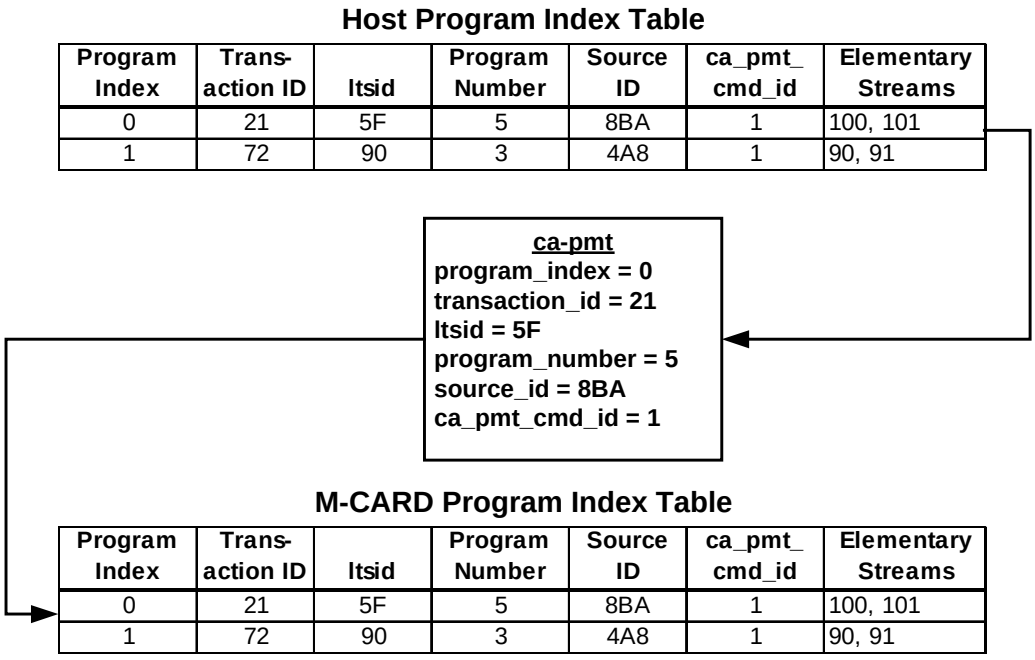


Figure 9.7-3 - Program Index Table 3

When a ***ca_pmt()*** APDU arrives with ca_pmt_cmd_id 0x01 (ok_descramble) or 0x03 (query), the M-CARD allocates the necessary CA resources and assigns them to the program index designated in the ***ca_pmt()*** APDU. Upon receipt of a ***ca_pmt()*** APDU containing a program_index for which resources have previously been assigned, the M-CARD SHALL release the former allocations. The M-CARD SHALL release any CA resources not required, such as any allocated to programs with no CA descriptors in the ***ca_pmt()*** APDU.

The M-CARD SHALL cease decryption of, and release CA resources assigned to, a program listed in a ca_pmt_cmd_id 0x04 (not_selected) command.

The Host assigns a transaction_id to each ***ca_pmt()*** APDU**.** The Host increments the transaction_id with every ***ca_pmt()*** APDU per program index it sends. The Host SHALL maintain a separate transaction_id counter for each program index, so that the transaction_ids increment independently for each program index in the ***ca_pmt()*** APDU.

The M-CARD SHALL use the transaction_id from the ***ca_pmt()*** APDU in any replies or updates it sends regarding that ***ca_pmt()*** APDU. This allows the host to match the M-CARD's messages with the corresponding ***ca_pmt()***. Obsolete replies or updates may then be discarded.

The Host MAY set the value of the Source ID field in the ***ca_pmt()*** APDU to 0 if this information is not available. If the Source ID field was previously set to 0 because it was not yet available, the Host SHALL update this field when the value becomes known in the next ***ca_pmt()*** APDU.

When a ***ca_pmt_reply()*** APDU is received with a ca_enable of "descrambling possible under conditions (purchase required)", the Host SHALL send the ***ca_pmt()*** APDU with a ca_pmt_cmd_id of "ok_descrambling". This will cause the M-CARD to allocate the necessary resources and assign them to the program index designated in the ca_pmt, so that the program can be descrambled immediately after the conditions are met. For example, when a ca_pmt(query) is received and the CableCARD determines it is an Impulse Pay-Per-View program, the ca_pmt_reply(descrambling possible under conditions (purchase required)) will be returned to the Host, and the Host will respond by sending a ca_pmt(ok_descrambling) to the CableCARD, to ensure it is prepared to descramble the program when the purchase is completed.

9.7.4 ca_pmt_reply

The Card SHALL send the ***ca_pmt_reply()*** APDU, as defined in Table 9.7-7 for S-Mode or Table 9.7-8 for M-Mode, to the Host after receiving a ***ca_pmt()*** APDU with ca_pmt_cmd_id 0x03 (query).

The M-CARD SHALL reply to ***ca_pmt()*** APDU with a ca_pmt_cmd_id 0x03 (query) and no ca_descriptors, by sending a ***ca_pmt_reply()*** APDU with ca_enable 0x01 (descrambling possible with no extra conditions).

The Card MAY send the ***ca_pmt()*** APDU after reception of a ***ca_pmt()*** APDU 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
ca_pmt_reply() {		
ca_pmt_reply_tag	24	uimsbf
length_field()		
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
CA_enable_flag	1	bslbf
if (CA_enable_flag == 1) {		
CA_enable /* 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_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.

- 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.

If the Card supports different authorizations for components of a program, 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.

Table 9.7-8 - M-Mode ca_pmt_reply() APDU Syntax (Resource Type 2 Version 1)

Syntax	No. of Bits	Mnemonic
ca_pmt_reply() {		
ca_pmt_reply_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_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 required). In order for the Card to descramble this program, the program must be purchased using a process specific to the Conditional Access System.
- 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, as defined in Table 9.7-9 for S-Mode and Table 9.7-10 for M-Mode, to inform the Host when CA information for the currently tuned program has changed. The Card SHALL always reference the service to which the Host is currently tuned in a ***ca_update()*** APDU. 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
ca_update() {		
ca_update_tag	24	uimsbf
length_field()		
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
CA_enable_flag	1	bslbf
if (CA_enable_flag == 1) {		
CA_enable /* 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_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

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.

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 required). In order for the Card to descramble this program, the program must be purchased using a process specific to the Conditional Access System.
- 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

9.8 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. The Host will perform its internal in-band tuning operation without requiring opening a session to the Host Control resource. Therefore, when it receives any in-band tuning APDUs from the Card, it can decide whether to grant them or not. Typically, unless either a Homing resource has been opened or the Host is in standby state or the CableCARD has transmitted a firmware_upgrade, the Host will be unable to grant the Card access to the in-band tuner.

The Host SHALL support a maximum of one session of the Host Control Resource, using the resource identifier(s) as defined in Table 9.8-1.

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 SHALL open a single session the Host Control Resource using the resource identifiers as defined in Table 9.8-1.

The Card SHALL leave the Host Control Resource open independent of the operation of the state of the Homing resource.

The Host Control resource consists of six 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

The Card SHALL use the ***OOB_TX_tune_req()*** APDU as defined in Table 9.8-3 to set up the Host's RDC 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 corresponds 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 corresponds 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 = Reserved
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 as defined in Table 9.8-7 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
  length_field()
  status_field
}
	
	24
	
	8
		
		uimsbf
		
		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. Otherwise, ***Status_field*** will be set to one of the following values:

0x00 Tuning granted
0x01 Tuning denied - RF transmitter not physically available
0x02 Tuning denied - RF transmitter busy
0x03 Tuning denied - Invalid parameters
0x04 Tuning denied - Other reasons
0x05-0xFF Reserved

The Card SHALL NOT attempt to perform any RDC operations after receiving an ***OOB_TX_tune_cnf()*** APDU with Status_field set to 0x01.

If the Host receives a ***OOB_TX_tune_req()*** APDU with any parameters that are out of range, it SHALL send the ***OOB_TX_tune_cnf()*** with Status_field set to 0x03.

9.8.3 OOB_RX_tune_req

The Card SHALL use the ***OOB_RX_tune_req()*** APDU as defined in Table 9.8-8 to set up the Host's FDC 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 as defined in Table 9.8-11 to the Card. The Host SHALL send the ***OOB_RX_tune_cnf()*** APDU 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`	`24`	`uimsbf`
`  length_field()`		
`  status_field`	`8`	`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 Card SHALL send the ***inband_tune_req()*** APDU as defined in Table 9.8-12 to request the Host to tune the inband QAM tuner. The APDU provides support for tuning to a source_id or a frequency with the modulation type.

Table 9.8-12 - inband_tune_req() APDU Syntax

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, indicates 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

9.8.6 inband_tune_cnf

When the Host receives an ***inband_tune_req()*** APDU, it SHALL respond with the ***inband_tune_cnf()*** APDU as defined in Table 9.8-14 for S-Mode and Table 9.8-15 for M-Mode.

Table 9.8-14 - S-Mode - inband_tune_cnf() APDU Syntax (Resource Type 1 Version 3)

Syntax	No. of Bits	Mnemonic

inband_tuning_cnf() {
  inband_tuning_cnf_tag
  length_field()
  tune_status
}
	24
	
	8
		uimsbf
		
		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 9.8-15 - M-Mode - inband_tune_cnf() APDU Syntax (Resource Type 1 Version 3)

Syntax	No. of Bits	Mnemonic

inband_tuning_cnf() {
inband_tuning_cnf_tag
  length_field()
  ltsid
  tune_status
}
	24
	
	8
	8
		uimsbf
		
		uimsbf
		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

If the Host responds to an ***inband_tuning_req()*** APDU with tune_status field = 0x00 (Tuning accepted) in the ***inband_tuning_cnf()*** APDU, the Host SHALL NOT remove any MPEG packets from the transport stream associated with the tuning request (i.e., the indicated transport stream is delivered to the M-Card intact).

9.9 Generic IPPV Support

The CableCARD Interface support of the Generic IPPV resource is deprecated.

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.

The Host SHALL support a maximum of one session to the System Time Support Resource using the identifier as defined in Table 9.10-1.

The Card SHALL open a maximum of one session to the System Time Support Resource using the identifier as defined in Table 9.10-1.

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

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 SHALL send the ***system_time_inq()*** APDU as defined in Table 9.10-3 to the Host.

Table 9.10-3 - Transmission of system_time_inq

Syntax	No. of Bits	Mnemonic
`system_time_inq () {`		
`  system_time_inq_tag`	24	uimsbf
`  length_field()`		
`  response_interval`	8	uimsbf
`}`		

system_time_inq_tag 0x9F8442

response_interval How often, in seconds, the Host should send the ***system_time()*** APDU to the Card, starting immediately. A value of 0x00 means that only a single ***system_time()*** APDU is to be sent, immediately.

9.10.2 system_time

The Host SHALL send the ***system_time()*** APDU as defined in Table 9.10-4 to the Card immediately in response to the ***system_time_inq()*** APDU.

If the response_interval in a ***system_time_inq()*** is non-zero, the Host SHALL send the ***system_time()*** APDU every response_interval seconds for the duration of the existence of the System Time session.

Table 9.10-4 - system_time APDU

Syntax	No. of Bits	Mnemonic

system_time() {
  system_time_tag
  length_field()
  system_time
  GPS_UTC_offset
}
	
	24
	
	32
	8
		
		uimsbf
		
		uimsbf
		uimsbf

system_time_tag 0x9F8443

system_time A 32-bit unsigned integer quantity representing the current system time as the number of GPS seconds since 12 AM, January 6, 1980, UTC.

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.

9.11 Man-Machine Interface (MMI)

The Host SHALL provide the MMI Resource using the identifier as defined in Table 9.11-1 with a maximum of one session. The Card SHALL only open one session to the MMI resource using the identifier as defined in Table 9.11-1 to initialize one or more MMI dialogs. The Host SHALL keep the MMI session open during normal operation. The Card SHALL keep the MMI session 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

When the Host is operating in M-Mode, and receives the ***open_mmi_req()*** APDU, the Host will display in a broadcast form the MMI dialog on all of the outputs. When the Host receives a ***close_mmi_cnf()*** the Host will close all MMI dialog messages it opened. If the Host sends the ***open_mmi_req()***, for support of Host diagnostics, the Host will use the dialog number to track what output the MMI dialog will be exchanged with.

If the Host supports the Full Screen Window Type, and if that Window Type is requested by the Card via the ***open_mmi_req()*** 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 ***open_mmi_req()*** 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 ***open_mmi_req()*** APDU, the Host SHALL present each subsequent MMI dialog, up to the maximum number conveyed, in an Overlay Window.

If multiple MMI Overlay Windows are supported, the Host SHALL support a method for navigating between the different windows.

Table 9.11-1 - MMI Support Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
MMI	S-Mode/M-Mode	64	2	1	0x00400081

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 as defined in Table 9.11-3 to the Host 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`	24	uimsbf
`  length_field()`		
`  display_type`	8	uimsbf
`  url_length`	16	uimsbf
`  for (i=0; i<url_length; i++) {`		
`    url_byte`	8	uimsbf
`  }`		
`}`		

open_mmi_req_tag 0x9F8820

display_type Describes how the MMI dialog takes 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, the Host SHALL reply with an ***open_mmi_cnf()*** APDU as defined in Table 9.11-4 to confirm the status of the request. When the Host is operating in M-Mode and receives an ***open_mmi_req()*** APDU, the Host SHALL open the MMI dialog on all of its outputs.

Table 9.11-4 - open_mmi_cnf

Syntax	No. of Bits	Mnemonic

open_mmi_cnf() {
  open_mmi_cnf_tag
  length_field()
  dialog_number
  open_status
}
	
	24
	
	8
	8
		
		uimsbf
		
		uimsbf
		uimsbf

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

9.11.3 close_mmi_req

The Card SHALL send a ***close_mmi_req()*** APDU as defined in Table 9.11-5 to the Host to close an MMI dialog previously opened with an ***open_mmi_req()*** APDU.

Table 9.11-5 - close_mmi_req

Syntax	No. of Bits	Mnemonic

close_mmi_req() {
  close_mmi_req_tag
  length_field()
  dialog_number
}
	
	24
	
	8
		
		uimsbf
		
		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 as defined in Table 9.11-6 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). After receiving a ***close_mmi_req()*** APDU from the Card, the Host SHALL close the specified MMI dialog.

Table 9.11-6 - close_mmi_cnf

Syntax	No. of Bits	Mnemonic

close_mmi_cnf() {
close_mmi_cnf_tag
length_field()
dialog_number
}
	24
	
	8
		uimsbf
		
		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 Host SHALL support the CableCARD Device Resources resource using the identifier as defined in Table 9.12-1. The M-CARD when operating in M-Mode SHALL open a session to the CableCARD Device Resources resource using the identifier as defined in Table 9.12-1 on the Host to indicate its multi-stream capabilities. The Card will use this resource to communicate the maximum number of transport streams it supports, how many programs it supports, and how many elementary streams it supports.

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.

If the Card informs the Host which PIDs it requires through the CableCARD Capability Discovery resource, CARD RES; the Host SHALL NOT remove those PIDs from each transport stream. Since the PIDS that the Card requires may change, it may send the ***request_pids_cnf()*** APDU at any time. Upon receipt of any ***request_pids_cnf()*** APDU, the Host SHALL update its list of PIDs to maintain in each transport stream. The Card MAY request a maximum of 8 non-program PIDs be transmitted to it per LTSID for internal consumption.

In the event that the Card is unable to meet the stream, program, or PID processing requirements that a user has requested, the Host SHALL 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	→

APDU Name	Tag Value	Resource	Direction Host ↔ Card
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

The Card SHALL send the ***stream_profile()*** APDU as defined in Table 9.12-3 to the Host to report the number of streams that it will support. This includes the maximum number of streams that the Card can support. The Card operating in M-Mode SHALL report at least 4 MPEG transport streams in the maximum_number_of_streams field in the ***stream_profile()*** APDU.

Table 9.12-3 - stream_profile APDU Syntax

Syntax	No. of bits	Mnemonic

stream_profile() {
  stream_profile_tag
  length_field()
  max_number_of_streams
}
	
	24
	
	8
		
		uimsbf
		
		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.

9.12.2 stream_profile_cnf APDU

When the Host receives the ***stream_profile()*** APDU from the Card, it SHALL respond with the ***stream_profile_cnf()*** APDU as defined in Table 9.12-4.

Table 9.12-4 - stream_profile_cnf APDU

Syntax	No. of bits	Mnemonic

stream_profile_cnf() {
  stream_profile_cnf_tag
  length_field()
  number_of_streams_used
}
	
	24
	
	8
		
		uimsbf
		
		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

The Card SHALL send the ***program_profile()*** APDU as defined in Table 9.12-5 to the Host to report the maximum number of simultaneous programs, summed across all transport streams, that the Card's CA system can simultaneously decrypt. The Card operating in M-Mode SHALL report the ability to decrypt at least 4 simultaneous programs in the maximum_number_of_programs field in the ***program_profile()*** APDU.

Table 9.12-5 - program_profile APDU

Syntax	No. of bits	Mnemonic

program_profile() {
  program_profile_tag
  length_field()
  max_number_of_programs
}
	
	24
	
	8
		
		uimsbf
		
		uimsbf

program_profile_tag Value = 0x9FA012

max_number_of_programs The maximum number of programs that the Card's CA system can simultaneously decrypt (must be greater than or equal to four).

9.12.4 program_profile_cnf APDU

When the Host receives the ***program_profile()*** APDU from the Card, it SHALL respond with the ***program_profile_cnf()*** APDU as defined in Table 9.12-6.

Table 9.12-6 - program_profile_cnf APDU

Syntax	No. of bits	Mnemonic

program_profile_cnf() {
  program_profile_cnf_tag
  length_field()
}
	24	uimsbf

program_profile_cnf_tag Value = 0x9FA013

9.12.5 es_profile APDU

The Card SHALL send the ***es_profile()*** APDU as defined in Table 9.12-7 to the Host to report the maximum number of simultaneous elementary streams, summed across all transport streams that the Card can support. The Card, operating in M-Mode, SHALL report the ability to manage at least 16 elementary streams in the maximum_number_of_es field in the ***es_profile()*** APDU.

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	No. of bits	Mnemonic

es_profile() {
  es_profile_tag
  length_field()
  max_number_of_es
}
	
	24
	
	8
		
		uimsbf
		
		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 16.

9.12.6 es_profile_cnf APDU

When the Host receives the ***es_profile()*** APDU from the Card, it SHALL respond with the ***es_profile_cnf()*** APDU as defined in Table 9.12-8.

Table 9.12-8 - es_profile_cnf APDU Syntax

Syntax	No. of bits	Mnemonic

es_profile_cnf() {
es_profile_cnf_tag
length_field()
}
	24	uimsbf

es_profile_cnf_tag Value = 0x9FA015

9.12.7 request_pids APDU

If the Host performs transport stream packet filtering (removal of packets corresponding to unneeded PIDs), the Host SHALL send the ***request_pids()*** APDU as defined in Table 9.12-9 to request a list of PIDs required by the Card.

If the Host performs transport stream filtering, then the Host SHALL NOT remove PID 0 (PAT), the PMT PID, ECM PIDs and any other Elementary Stream PIDs that require Card decryption.

Please refer to Section 9.8.6 regarding filtering restrictions for Card requested transport streams using the ***inband_tuning_req()*** APDU.

Table 9.12-9 - request_pids APDU

Syntax	No. 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 with pid_filtering_status = 0x01 by sending the ***request_pids_cnf()*** APDU as defined in Table 9.12-10 with the non-program PIDs it requires. The Card MAY send the ***request_pids_cnf()*** APDU at any time.

Table 9.12-10 - request_pids_cnf APDU

Syntax	No. of bits	Mnemonic
request_pids_cnf() {		
request_pids_cnf_tag	24	uimsbf
length_field()		
ltsid	8	uimsbf
number_of_pids	8	uimsbf
for (i=0; i<number_of_pids; i++) {		
zero	3	uimsbf
pid	13	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 non-program PIDs required; maximum is 8

zero 3 bits of zero

pid PID value

9.13 Copy Protection

A Copy Protection Resource is opened by the Card as defined in [CCCP] and complies to the interface requirement specifications as defined in [CCCP].

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	See [CCCP] for the current Resource Class, Type and Version for this resource.			

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 Host SHALL provide the Extended Channel Support resource using identifier(s) defined in Table 9.14-1 to register the data flows to send and receive data over the Extended Channel. The Card SHALL support the Extended Channel Support resource using identifier(s) as defined in Table 9.14-1.

All Hosts are required to provide the hardware necessary to support a QPSK downstream (FDC) channel for the Card. Host 2.0 devices are required to incorporate a QPSK upstream (RDC) channel for the Card and a DOCSIS embedded Cable Modem (eCM) for bidirectional IP support. The eCM is required to support the DOCSIS Set-top Gateway (DSG) function. There are four types of flows on the extended channel: MPEG, IP, Socket, and DSG.

When in SCTE 55 mode, the Card will forward data formatted as MPEG sections 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 FDC channel or the DSG flows 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 three different ways of using the Extended Channel Support resource:

- The application is in the Host and the data is transferred to/from the headend via the QPSK modem.
- The application is in the Card and the data is transferred to/from the headend via the Host's eCM.
- The application is in the Host and the data is transferred to/from the headend via the Host's eCM. For example, in DSG mode, SI data is transferred from the eCM to the Card and then from the Card to the Host in an MPEG section flow (as explained in Section 9.14.1).

After a flow is lost, i.e., after a ***lost_flow_ind()*** is sent by the link device and a ***lost_flow_cnf()*** with the same flow_id is sent by the application, the flow does not exist anymore.

After a flow is deleted, i.e., after a ***delete_flow_req()*** is sent by the application and a ***delete_flow_cnf()*** with the same flow_id is sent by the link device, the flow does not exist anymore.

An application that needs to make use of a flow that doesn't exist, either because the link device lost the flow or because the application deleted the flow, SHALL request the flow again using ***new_flow_req()***.

If the link device granted an IP_U flow and the IP address provided in the ***new_flow_cnf()*** changes, the link device SHALL send a ***lost_flow_ind()*** for that IP_U flow.

Version 1 of this resource is required for Hosts that do not have an embedded High Speed Host (DOCSIS) Modem. Versions 2 and higher of this resource are required for Hosts that have an embedded High Speed Host (DOCSIS) Modem.

Messaging for versions 2, 3, 4 and 5 of this resource are defined in Annex E of this specification.

Version 5 of this resource adds service_type = 0x04 to the ***new_flow_req()*** APDU.

Version 5 of this resource removes the following APDUs:

- inquire_DSG_mode()
- set_DSG_mode()
- DSG_error()
- DSG_message()
- configure_advanced_dsg()
- send_DCD_info()

When version 5 or higher of the Extended Channel Resource is implemented, the Host SHALL implement the DSG Resource, as defined in Section 9.20, for DSG applications.

When version 5 or higher of the Extended Channel Resource is supported, the Host SHALL use the APDUs defined for the DSG resource instead of the DSG-related APDUs previously defined for the Extended Channel Resource.

When version 5 or higher of the Extended Channel Resource is supported, the Card SHALL use the APDUs defined for the DSG resource instead of the DSG-related APDUs previously defined for the Extended Channel Resource.

If the Host supports only version 1, 2, 3, and/or 4 of the Extended Channel Resource, the Card SHALL NOT request to open the DSG Resource.

If the Host supports only version 1, 2, 3, and/or 4 of the Extended Channel Resource, the Card SHALL use the Extended Channel Resource for DSG messaging as defined in Annex E.7.

Version 6 adds DHCP information to the socket service_type and extends the IP address length to 128 bits to support IPv6.

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
Extended Channel Support	S-Mode/M-Mode	160	1	5	0x00A00045
Extended Channel Support	S-Mode/M-Mode	160	1	6	0x00A00046

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

* The DSG-related APDUs are not used in Version 5 or higher of the Extended Channel Resource and are redefined in the DSG resource. For versions 2-4, definitions of these DSG-related APDUs are found in Annex E.

9.14.1 new_flow_req APDU

The Host SHALL issue a ***new_flow_req()*** APDU as defined in Table 9.14-3 to register a new flow with the Card.

The Card SHALL issue a ***new_flow_req()*** APDU as defined in Table 9.14-3 to register a new flow with the Host.

Any device opening a flow that is listed as N/A or not listed in the table (such as the Card requesting an IP_U flow when in QPSK mode) will receive a status_field = 0x02, service type not available.

In any DSG mode, the Card SHALL request to open a single flow using the ***new_flow_req()*** APDU with DSG as the service_type.

When not in Advanced DSG mode, the Host SHALL open an MPEG section flow with the SI_Base PID (0x1FFC) to the Card for reception of. SCTE 65 SI messages, SCTE 18 EAS messages, CVTs and OCAP XAITs over the Extended Channel.

When the Host is operating in Advanced DSG mode and makes a request to open an MPEG flow to PID 0x1FFC over the Extended Channel, the Card MAY deny the request.

The service types available are MPEG section, IP unicast, IP multicast, Socket, and DSG.

The Host SHALL support at least six concurrent MPEG section service_type flows over the Extended Channel.

The Card SHALL support at least six concurrent MPEG section service_type flows over the Extended Channel.

When version 5 or higher of the Extended Channel Resource is used, the Host SHALL provide support for at least eight Socket service_type flows as requested by the Card.

When version 5 or higher of the Extended Channel Resource is used, the Card MAY request a Socket service_type flow.

When version 5 or higher of the Extended Channel Resource is used, the Card SHALL limit the Socket service_type flows to a maximum of 8.

The Host SHALL support one DSG service_type flow over the Extended Channel.

The Card SHALL support one DSG service_type flow over the Extended Channel.

The Host SHALL support at least one IP Unicast (IP_U) service_type flow, providing support for UDP and/or TCP protocols over the Extended Channel.

The Card SHALL support at least one IP Unicast (IP_U) service_type flow, providing support for UDP and/or TCP protocols over the Extended Channel.

The devices on either side of the CHI are required to support only one outstanding ***new_flow_req()*** transaction at a time.

The following are different types of service flows used by the devices on either side of the CHI:

MPEG section - This service type is applicable only for flows between the Card and the Host. The Card SHALL provide the requested MPEG service flow across the Extended Channel in the form of MPEG sections (both long and short form). This type of flow is unidirectional, from Card to Host only. The Card SHALL provide MPEG sections across the Extended Channel with the value of the section_length_field not to exceed 4,093.

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 MPEG 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 MPEG long-form messages that pass the CRC check will be forwarded to the Host. The Card SHALL discard MPEG table sections that are incomplete or fail the CRC check.

The 32-bit CRC may 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. The Host SHALL be responsible for CRC validation of short-form MPEG sections received over the Extended Channel.

IP Unicast - This service type applies for flows between the Host and an SCTE 55 modem in the Card (SCTE 55 mode). The requested flow will be in the form of IP packets addressed to or from the Host's IP address when in SCTE 55 mode. The IP Unicast flow may be bidirectional. The Card SHALL support a maximum total length of any IP_Unicast packet in SCTE 55 mode of 1,500 bytes. The Host SHALL support a maximum total length of any IP_Unicast packet in SCTE 55 mode of 1500 bytes.

IP Multicast - This service type is applicable for flows between the Host and a modem in the Card. The requested flows will be in the form of multicast IP packets addressed to the multicast_group_ID assigned IP address. This type of flow is unidirectional, from link device to non-link device. The maximum total length of any IP packet is expected to be 1,500 bytes.

DSG - This service type applies to unidirectional flows of data from the Host to the Card. This type of flow is unidirectional, from Host to Card only.

Socket - This service type is only applicable when the Host and the Card are in DSG mode.

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
}		
if(service_type == 04) { /* Socket*/		
protocol_flag	8	uimsbf
local_port_number	16	uimsbf
remote_port_number	16	uimsbf
remote_address_type	8	uimsbf
/* Use of 0x00 is deprecated */		
if(remote_address_type==0x00) {		
name_length	8	uimsbf
for(int i=0;i<name_length;++i) {		
name_byte	8	uimsbf
}		
}		
if(remote_address_type == 0x01)		
ipv4_address	32	uimsbf
if(remote_address_type == 0x02)		
ipv6_address	128	uimsbf
connection_timeout	8	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 Socket
0x05-0xFF 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 over the Extended Channel.

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 device requesting the new flow SHALL NOT use the "end option" (code 255) in the option_byte field of the ***new_flow_req()*** APDU, 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 is responsible for filtering arriving multicast IP packets and delivering only packets matching the given multicast_group_ID address in an IP Multicast flow on the Extended Channel.

protocol_flag The type of socket flow requested.

- 00 UDP - instructs the host to establish a UDP socket for the Card to use to pass traffic on this flow.
- 01 TCP - instructs the host to establish a TCP socket for the Card to use to pass traffic on this flow.

local_port_number The local port number for a socket connection. This field MAY be 0.

remote_port_number The port number of the socket on the remote Host. For UDP Socket Flows, it is optional for Host to filter packets to the Card base on remote_port_number. That is, the Host may forward incoming UDP packets whose source port does not match the remote port specified in the ***new_flow_req()*** APDU.

remote_address_type The remote Host's IP address format.

- 00 name - DNS will be required to look up the remote Host's IP address (Deprecated)
- 01 ipv4 - 32-bit IPv4 address
- 02 ipv6 - 128-bit IPv6 address

name_length The number of bytes in the following for loop. (Deprecated)

name_byte These bytes specify the remote Host's name in ASCII format. This field may specify either a Host name or a fully qualified domain name. (Deprecated)

remote_IP_address The IP address of the remote Host. This address is in standard network-byte order so the higher order bits are sent first.

connection_timeout Number of seconds the Host will attempt to establish a TCP connection.

9.14.2 new_flow_cnf APDU

The Host SHALL return a ***new_flow_cnf()*** APDU as defined in Table 9.14-4 after receiving a ***new_flow_req()*** APDU.

The Card SHALL return a ***new_flow_cnf()*** APDU as defined in Table 9.14-4 after receiving a ***new_flow_req()*** APDU.

The Host and Card are required to support only one outstanding new_flow_req transaction at a time. If an additional ***new_flow_req()*** APDU is received while one is being processed, the Host SHALL send a ***new_flow_cnf()*** APDU with a status field of 0x04 (Network Busy).

If an additional ***new_flow_req()*** APDU is received while one is being processed, the Card SHALL send a ***new_flow_cnf()*** APDU with a status field of 0x04 (Network Busy).

Table 9.14-4 - 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	128	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
}		
link_IP_address	128	uimsbf
}		
if (service_type == Socket) {		
reserved	3	uimsbf
max_pdu_size	13	uimsbf
link_IP_address	128	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 Request denied, DNS not supported
- 0x07 Request denied, DNS lookup failed
- 0x08 Request denied, local port already in use or invalid
- 0x09 Request denied, could not establish TCP connection
- 0x0A Request denied, IPv6 not supported
- 0x0B-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. To avoid conflicts between the assignment of flow_ids between the Card and the Host, the Card and the Host will assign Ids in different ranges. The flow id value of 0x000000 is reserved and should not be assigned by either the Host or the Card.

The Card SHALL assign Extended Channel flow_ids in the range of 0x000001 to 0x7FFFFF in the ***new_flow_cnf()*** APDU.

The Host SHALL assign Extended Channel flow_ids in the range of 0x800000 to 0xFFFFFF in the ***new_flow_cnf()*** APDU.

service_type The requested service_type received in the ***new_flow_req()*** APDU.

IP_address The 128-bit IP address associated with the requested flow. If the address is IPv4, then the upper 96 bits SHALL be set to zero.

link_IP_address The 128-bit IP address assigned to the link device. If the address is IPv4, then the upper 96 bits SHALL be set to zero.

flow_type This field is not supported in any version of the extended channel resource.

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 9.14-5.

Bit 0 no_frag
bits 2:1 reserved

Table 9.14-5 - Flag field definitions

BITS		
2	1	0
reserved		no_frag

no_frag A 1-bit Boolean that designates if the network supports fragmentation. A value of 0_2 indicates that fragmentation is supported. A value of 1_2 indicated that fragmentation is not supported.

max_pdu_size A 13-bit unsigned integer number that designates the maximum PDU length that may be transmitted across the interface.

option_field_length An 8-bit unsigned integer number that represents the number of bytes of option field data to follow.

option_byte If service_type == IP_U, these bytes correspond to the DHCP options requested in the ***new_flow_req()*** message. If service_type == Socket, these bytes correspond to the DHCP options requested by the link device. The format of the field is as defined in [RFC2132].

The device replying with the ***new_flow_cnf()*** APDU SHALL NOT use the "end option" (code 255) in the option_byte field.

9.14.3 delete_flow_req APDU

The Host SHALL send the ***delete_flow_req()*** APDU as defined in Table 9.14-6 to the Card to delete a registered data flow over the Extended Channel.

The Card SHALL send the ***delete_flow_req()*** APDU as defined in Table 9.14-6 to the Host to delete a registered data flow over the Extended Channel.

Table 9.14-6 - delete_flow_req APDU Syntax

Syntax	No. of Bits	Mnemonic

delete_flow_req() {
  delete_flow_req_tag
  length_field()
  flow_id
}
	
	24
	
	24
		
		uimsbf
		
		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 Host receives a ***delete_flow_req()*** APDU, it SHALL respond with the ***delete_flow_cnf()*** APDU as defined in Table 9.14-7.

When the Card receives a ***delete_flow_req()*** APDU, it SHALL respond with the ***delete_flow_cnf()*** APDU as defined in Table 9.14-7.

Table 9.14-7 - delete_flow_cnf APDU Syntax

Syntax	No. of Bits	Mnemonic

delete_flow_cnf() {
delete_flow_cnf_tag
length_field()
flow_id
status_field
}
	
	24
	
	24
	8
		
		uimsbf
		
		uimsbf
		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

The Host SHALL indicate that a registered data flow has been lost by issuing the ***lost_flow_ind()*** APDU as defined in Table 9.14-8.

The Card SHALL indicate that a registered data flow has been lost by issuing the ***lost_flow_ind()*** APDU as defined in Table 9.14-8.

Table 9.14-8 - lost_flow_ind APDU Syntax

Syntax	No. of Bits	Mnemonic

lost_flow_ind() {
  lost_flow_ind_tag
  length_field()
  flow_id
  reason_field
}
	
	24
	
	24
	8
		
		uimsbf
		
		uimsbf
		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 Remote TCP socket closed
0x05 Socket read error
0x06 Socket write error
0x07-0xFF Reserved

9.14.6 lost_flow_cnf APDU

The Host SHALL respond with the ***lost_flow_cnf()*** APDU as defined in Table 9.14-9 when a ***lost_flow_ind()*** APDU is received.

The Card SHALL respond with the ***lost_flow_cnf()*** APDU as defined in Table 9.14-9 when a ***lost_flow_ind()*** APDU is received.

Table 9.14-9 - lost_flow_cnf APDU Syntax

Syntax	No. of Bits	Mnemonic

lost_flow_cnf() {
  lost_flow_cnf_tag
  length_field()
  flow_id
  status_field
}
	
	24
	
	24
	8
		
		uimsbf
		
		uimsbf
		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

9.15 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. The Card opens only one Generic Feature Control session to the Host and should never close the session.

9.15.1 Parameter Storage

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 queries 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 sends its confirmation of the feature list to the Host (feature_list_cnf). The Host then queries the Card to determine which generic features are supported in the Card and the headend (feature_list_req). The Card sends its feature list to the Host (feature_list) to which the Host then sends its confirmation (feature_list_cnf). This is called the generic feature list exchange.

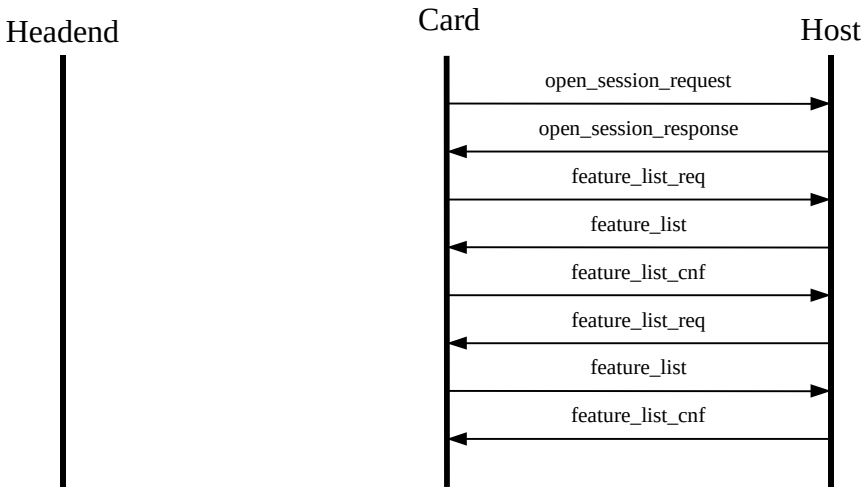


Figure 9.15-1 - Generic Feature List Exchange

If the generic feature list on the Host or the Card changes, then the changed device sends a ***feature_list_changed()*** APDU to the other device. The other device then performs the generic feature list exchange to obtain the new list.

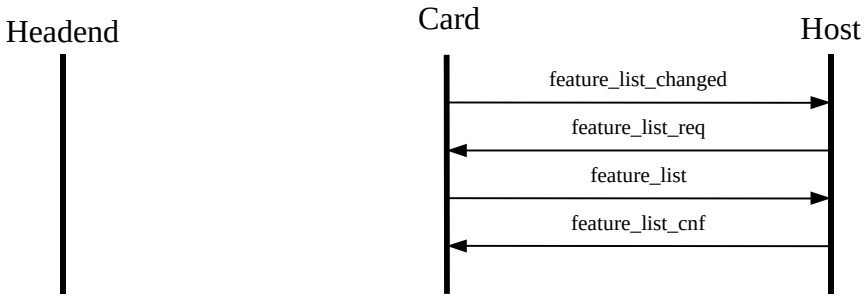


Figure 9.15-2 - Card Feature List Change

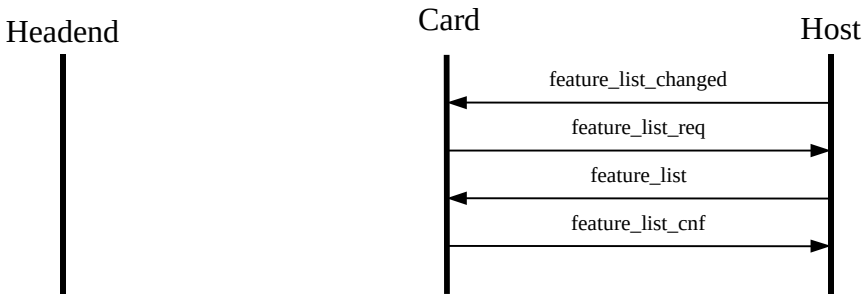


Figure 9.15-3 - Host Feature List Change

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 sends to the Card the parameters for all the generic features in the Host's generic feature list (feature_parameters). The Card replies with the confirmation (feature_parameters_cnf). The Card may utilize these generic feature parameters, transfer them to the headend, or ignore them.

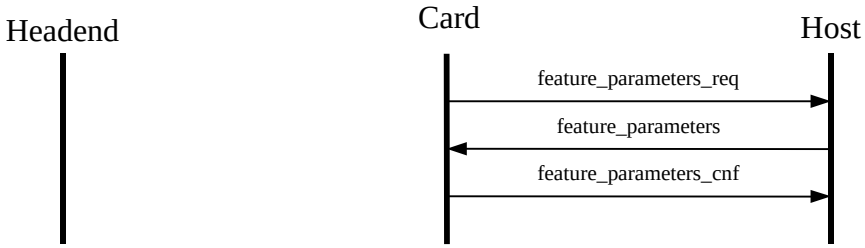


Figure 9.15-4 - Host to CableCARD Device Feature Parameters

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 sends these new parameters to the Card (feature_parameters). The Card replies with the confirmation (feature_parameters_cnf).

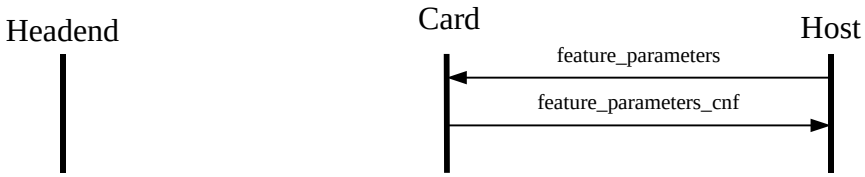


Figure 9.15-5 - Host Parameter Update

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 by sending a ***feature_parameters_req()*** APDU.

9.15.2.3 Headend to Host

It is not intended that the headend would send all the generic feature's parameters periodically. Most of the parameters would only be sent once at the request of the user or for staging of the device. The generic feature's parameters that may need to be sent periodically 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 transfers this information to the Host (feature_parameters). The Host will replace its parameters with the values in the APDU. If the Card utilizes the parameters, it will replace its internal parameters with the values in the message from the headend. The Host will respond with the confirmation (feature_parameters_cnf). The Host may receive parameters for generic features, which it does not support. The Host ignores any generic feature parameters that it does not implement.

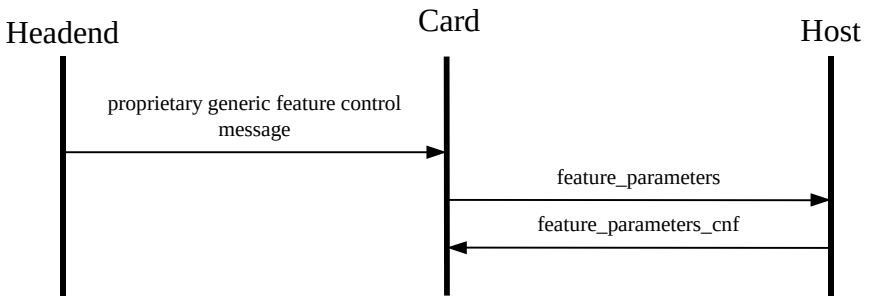


Figure 9.15-6 - Headend to Host Feature Parameters

9.15.3 Generic Feature Control Resource Identifier

The Host SHALL support the Generic Feature Control Resource using the resource identifier(s) as defined in Table 9.15-1.

The Card SHALL open only one session to the Generic Feature Control Resource using the resource identifier as defined in Table 9.15-1. Generic Feature Control type 1 version 2 and higher are optional for S-Mode.

Table 9.15-1 - Generic Feature Control Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic Feature Control	S-Mode/M-Mode	42	1	4	0x002A0044

9.15.4 Feature ID

The Host SHALL utilize the unique IDs for each supported generic feature as defined in Table 9.15-2

The Card SHALL utilize the unique IDs for each supported generic feature in Table 9.15-2.

The following is a list of the features and their assigned feature ID.

Table 9.15-2 - Feature Ids

Feature ID	Feature	Lowest Applicable Version
0x00	Reserved	
0x01	RF Output Channel	1
0x02	Parental Control PIN	1
0x03	Parental Control Settings	1
0x04	Purchase PIN	1
0x05	Time Zone	1
0x06	Daylight Savings Control	1
0x07	AC Outlet	1
0x08	Language	1
0x09	Rating Region	1
0x0A	Reset PINS	1
0x0B	Cable URL	1
0x0C	EAS location code	1
0x0D	VCT ID	3
0x0E	Turn-on Channel	3
0x0F	Terminal Association	4
0x10	Download Group-ID	4
0x11	Zip Code	4
0x12-0x6F	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

After the session to the Generic Feature Control Resource is opened, the Card SHALL send the ***feature_list_req()*** APDU as defined in Table 9.15-4 to the Host to query the generic features that are supported.

The Host SHALL send the ***feature_list_req()*** APDU as defined in Table 9.15-4 to the Card to query the generic features that are supported in the Card.

Table 9.15-4 - feature_list_req APDU Syntax

Syntax	No. of Bits	Mnemonic

feature_list_req() {
  feature_list_req_tag
  length_field()
}
	24	uimsbf

feature_list_req_tag 0x9F9802

9.15.5.2 feature_list APDU

After receiving the ***feature_list_req()*** APDU, the Host SHALL send the ***feature_list()*** APDU as defined in Table 9.15-5 to the Card. The Host SHALL include in the list all the features that it is capable of sending to the Card in ***feature_parameters()*** APDU(s). The Host SHALL NOT send any features in ***feature_parameters()*** APDU that it does not list in ***feature_list()*** APDU.

After receiving the ***feature_list_req()*** APDU, the Card SHALL send the ***feature_list()*** APDU as defined in Table 9.15-5 to the Host. The Card SHALL include in the list all the features that it is capable of sending to the Host in ***feature_parameters()*** APDU(s). The Card SHALL NOT send any features in ***feature_parameters()*** APDU that it does not list in ***feature_list()*** APDU.

Table 9.15-5 - feature_list APDU Syntax

Syntax	No. of Bits	Mnemonic

feature_list() {
  feature_list_tag
  length_field()
  number_of_features
  for (i=0; i<number_of_features; i++) {
  feature_id
  }
}
	24
	
	8
	
	8
		uimsbf
		
		uimsbf
		
		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 SHALL respond with the ***feature_list_cnf()*** APDU as defined in Table 9.15-6.

After receiving the ***feature_list()*** APDU, the Card SHALL respond with the ***feature_list_cnf()*** APDU as defined in Table 9.15-6.

Table 9.15-6 - feature_list_cnf APDU Syntax

Syntax	No. of Bits	Mnemonic

feature_list_cnf() {
  feature_list_cnf_tag
  length_field()
}
	24	uimsbf

feature_list_cnf_tag 0x9F9804

9.15.5.4 feature_list_changed APDU

The Host SHALL send the ***feature_list_changed()*** APDU as defined in Table 9.15-7 to inform the Card that its feature list has changed.

The Card SHALL send the ***feature_list_changed()*** APDU as defined in Table 9.15-7 to inform the 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 as defined in Table 9.15-8 to the Host. The Host does 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 as defined in Table 9.15-9 with its feature list to the Card after receiving a ***feature_parameters_req()*** APDU.

The Host SHALL send the ***feature_parameters()*** APDU as defined in Table 9.15-9 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 received in the ***feature_parameters()*** APDU that it does not support.

The Card SHALL send the ***feature_parameters()*** APDU as defined in Table 9.15-10 to the Host as soon as both the generic feature list exchange has occurred and the Card has received a message from the headend containing generic feature parameters.

The Card SHALL send the ***feature_parameters()*** APDU as defined in Table 9.15-10 to the Host every time it receives a message from the headend changing one or more feature parameters.

Upon receipt of the ***feature_parameters()*** APDU from the Card, the Host SHALL update its parameters with the values in the APDU.

Table 9.15-9 - feature_parameters APDU Syntax (Type 1 Versions 1-3)

Syntax	No. of Bits	Mnemonic
feature_parameters() {		
feature_parameters_tag	24	uimsbf
length_field()		
number_of_features	8	uimsbf
for (i=0; i<number_of_features; i++) {		
feature_id	8	uimsbf
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()		
}		
if (feature_id == 0x0D) {		
vct_id()		
}		
if (feature_id == 0x0E) {		
turn_on_channel()		
}		
}		
}		

feature_parameters_tag	0x9F9807
number_of_features	Number of features to report
feature_id	Assigned feature ID number (see Table 9.15-2)
rf_output_channel	RF output channel
p_c_pin	Parental Control PIN parameter
p_c_settings	Parental Control Settings parameter
ippv_pin	IPPV PIN parameter
time_zone	Time Zone parameter *This feature is only utilized if the cable system crosses time zones.*
daylight_savings	Daylight Savings parameter *This feature is only utilized if the cable system encompasses both areas, which recognize daylight savings and those which do not.*
ac_outlet	AC Outlet parameter
language	Language parameter
rating_region	Rating Region parameter
reset_pin	Reset PINs
cable_urls	URL list
ea_location_code	EAS location code
vct_id	VCT ID
turn_on_channel	Turn-on virtual channel

Table 9.15-10 - feature_parameters APDU Syntax (Type 1 Version 4)

Syntax	No. of Bits	Mnemonic
`feature_parameters() {`		
`  feature_parameters_tag`	24	uimsbf
`  length_field()`		
`  number_of_features`	8	uimsbf
`  for (i=0; i<number_of_features; i++) {`		
`    feature_id`	8	uimsbf
`    length`	16	uimsbf
`    feature_parameter_data()`		
`  }`		
`}`		

feature_parameters_tag	0x9F9807
number_of_features	Number of features to report
feature_id	Assigned feature ID number (see Table 9.15-2)
length	The number of bytes to follow
feature_parameter_data()	Each feature parameter structure is described in the following sections starting from Section 9.15.5.7.1

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 Host receives the ***feature_parameters()*** APDU, it SHALL respond with the feature_parameters_cnf() APDU as defined in Table 9.15-11.

When the Card receives the ***feature_parameters()*** APDU, it SHALL respond with the ***feature_parameters_cnf()*** APDU as defined in Table 9.15-11.

Table 9.15-11 - Feature Parameters Confirm Object Syntax

Syntax	No. of bits	Mnemonic

Feature_parameters_cnf() {
  feature_parameters_cnf_tag
  length_field()
  number_of_features
  for(i=0; i<number_of_features; i++){
  feature_id
  status
  }
}
	
	24
	
	8
	
	8
	8
		
		uimsbf
		
		uimsbf
		
		uimsbf
		uimsbf

feature_parameters_tag Value = 0x9F9808

number_of_features Number of features to report

feature_ID Assigned feature ID number as defined in Table 9.15-2

status 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

The Host SHALL support the RF Output Channel feature as defined in Table 9.15-12.

The Card SHALL support the RF Output Channel feature as defined in Table 9.15-12.

Table 9.15-12 - 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 RF output channel value that it cannot accommodate in a Generic Feature parameter and will use its previous value.

output_channel_ui Enable RF output channel user interface.

00 Reserved
01 Enable RF output channel user interface

02 Disable RF output channel user interface
03-0xFF Reserved

If the output_channel_ui parameter is set to disable the RF output channel user interface, the Host SHALL disable the user from changing the RF output channel.

9.15.5.7.2 p_c_pin

If the p_c_pin feature is supported, the Host SHALL support the p_c_pin parameters as defined in Table 9.15-13.

If the p_c_pin feature is supported, the Card SHALL support the p_c_pin parameters as defined in Table 9.15-13.

Table 9.15-13 - 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

If the p_c_settings feature is supported, the Host SHALL support the p_c_pin parameters as defined in Table 9.15-14.

If the p_c_settings feature is supported, the Card SHALL support the p_c_pin parameters as defined in Table 9.15-14.

Table 9.15-14 - 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.

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.

The Host SHALL specify two-part or one-part channel numbers as defined in [SCTE65] for the major_channel_number and the minor_channel_number parameters in the p_c_settings feature.

The Card SHALL specify two-part or one-part channel numbers as defined in [SCTE65] or the major_channel_number and the minor_channel_number parameters in the p_c_settings feature.

9.15.5.7.4 purchase_pin

If the purchase_pin feature is supported, the Host SHALL support the purchase_pin parameters as defined in Table 9.15-15.

If the purchase_pin feature is supported, the Card SHALL support the purchase_pin parameters as defined in Table 9.15-15.

Table 9.15-15 - purchase_pin

Syntax	No. of Bits	Mnemonic

purchase_pin() {
  purchase_pin_length
  for (i=0; i<purchase_pin_length; i++) {
  purchase_pin_chr
  }
}
	8
	
	8
		uimsbf
		
		uimsbf

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

If the time_zone feature is supported, the Host SHALL support the time_zone feature parameters as defined in Table 9.15-16.

If the time_zone feature is supported, the Card SHALL support the time_zone feature parameters as defined in Table 9.15-16.

Table 9.15-16 - time_zone

Syntax	No. of Bits	Mnemonic

time_zone() {
  time_zone_offset
}
	16	tcimsbf

time_zone_offset Two's complement integer offset, in number of minutes, from UTC. The value represented will be in the range of -12 to +12 hours.

9.15.5.7.6 daylight_savings

If the daylight_savings feature is supported, the Host SHALL support the daylight_savings feature parameters as defined in Table 9.15-17 if Type 1, Version 1 of the Generic Features Resource is supported, or Table 9.15-18 for Type 1, Version 2 and above of the Generic Features Resource.

If the daylight_savings feature is supported, the Card SHALL support the daylight_savings feature parameters as defined in Table 9.15-17 if Type 1, Version 1 of the Generic Features Resource is supported, or Table 9.15-18 for Type 1, Version 2 and above of the Generic Features Resource.

Table 9.15-17 - daylight_savings (Type 1 Version 1)

Syntax	No. of Bits	Mnemonic

daylight_savings() {
  daylight_savings_control
}
	8	uimsbf

daylight_savings_control Daylight savings time control. The Card derives this information from the configuration messages received from the Headend.

0x00 Ignore this field
0x01 Do not use daylight savings time
0x02 Use daylight savings
0x03-0xFF Reserved

Table 9.15-18 - daylight_savings (Type 1 Version 2 and above)

Syntax	No. of Bits	Mnemonic

daylight_savings() {
  daylight_savings_control
  if(daylight_savings_control == 0x02) {
  daylight_savings_delta
  daylight_savings_entry_time
  daylight_savings_exit_time
  }
}
	8
	
	8
	32
	32
		uimsbf
		
		uimsbf
		uimsbf
		uimsbf

daylight_savings_control Daylight savings time control. The Card derives information needed to build the daylight_savings() feature from the configuration messages received from the Headend.

0x00 Reserved
0x01 Do not use daylight savings time
0x02 Use daylight savings
0x03-0xFF Reserved

daylight_savings_delta Daylight savings delta time in number of minutes.

daylight_savings_entry_time Daylight savings entry time given as system_time value as defined in [SCTE65] prior to any local time conversion. For example, "3/14/2010 02:00:00" would be transmitted as 0x38C705AF.

Note: This hex value includes a +15 sec GPS offset, which is the current value for the year 2010, but may change in the future.

daylight_savings_exit_time Daylight savings exit time given as system_time value as defined in [SCTE65] prior to any local time conversion. The value of this parameter shall be greater

than the value of daylight_savings_entry_time. For example, "11/07/2010 02:00:00" would be transmitted as 0x3A00CAAF.

Note: This hex value includes a +15 sec GPS offset which is the current value for the year 2010 but may change in the future.

If the GPS_UTC_offset in the [SCTE65] System Time Table is equal to zero, then the system_time field carries UTC time directly and the current offset must be subtracted from daylight_savings_entry_time and daylight_savings_exit_time.

If the GPS_UTC_offset in the [SCTE65] System Time Table is non-zero, then the system_time field can be compared directly to daylight_savings_entry_time and daylight_savings_exit_time.

9.15.5.7.7 ac_outlet

If the ac_outlet feature is supported, the Host SHALL support the ac_outlet feature parameters as defined in Table 9.15-19.

If the ac_outlet feature is supported, the Card SHALL support the ac_outlet feature parameters as defined in Table 9.15-19.

Table 9.15-19 - 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

If the language feature is supported, the Host SHALL support the language feature parameters as defined in Table 9.15-20.

If the language feature is supported, the Card SHALL support the language feature parameters as defined in Table 9.15-20.

Table 9.15-20 - 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

If the rating_region feature is supported, the Host SHALL support the rating_region feature parameters as defined in Table 9.15-21.

If the rating_region feature is supported, the Card SHALL support the rating_region feature parameters as defined in Table 9.15-21.

Table 9.15-21 - 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

If the reset_pin feature is supported, the Host SHALL support the reset_pin feature parameters as defined in Table 9.15-22.

If the reset_pin feature is supported, the Card SHALL support the reset_pin feature parameters as defined in Table 9.15-22.

Table 9.15-22 - reset_pin

Syntax	No. of Bits	Mnemonic

reset_pin() {
  reset_pin_control
}
	8	uimsbf

reset_pin_control Defines the control of resetting PIN(s).

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

If the cable_urls feature is supported, the Host SHALL support the cable_urls feature parameters as defined in Table 9.15-23, using the restricted set of characters and the generic syntax for the url_char parameter as defined in [RFC2396], “Uniform Resource Identifier (URI): Generic Syntax”.

If the cable_urls feature is supported, the Card SHALL support the cable_urls feature parameters as defined in Table 9.15-23, using the restricted set of characters and the generic syntax for the url_char parameter as defined in [RFC2396], “Uniform Resource Identifier (URI): Generic Syntax”.

Table 9.15-23 - cable_urls

Syntax	No. of Bits	Mnemonic
cable_urls() {		
number_of_urls	8	uimsbf
for (i=0; i<number_of_urls; i++) {		
url_type	8	uimsbf
url_length	8	uimsbf
for (j=0; j<url_length; j++) {		
url_char	8	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

url_length Length of the URL. Used in the following for loop. The maximum length is 255 bytes.

url_char A URL character. The restricted set of characters and generic syntax defined in [RFC2396], "Uniform Resource Identifier (URI): Generic Syntax", will be used.

9.15.5.7.12 EA_location_code

If the EA_location_code feature is supported, the Host SHALL support the EA_location_code feature parameters as defined in Table 9.15-24.

If the EA_location_code feature is supported, the Card SHALL support the EA_location_code feature parameters as defined in Table 9.15-24.

Table 9.15-24 - 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]

9.15.5.7.13 vct_id

If the vct_id feature is supported, the Host SHALL support the vct_id feature parameters as defined in Table 9.15-25.

If the vct_id feature is supported, the Card SHALL support the vct_id feature parameters as defined in Table 9.15-25.

Table 9.15-25 - VCT ID

Syntax	No. of Bits	Mnemonic

vct_id() {
  vct_id
}
	16	uimsbf

vct_id VCT ID value to use

9.15.5.7.14 Turn-on Channel

This generic feature carries the number of the virtual channel that a Host tunes to upon power up. This generic feature may optionally be implemented in the S-Mode, whereas it is required in the M-Mode.

If the turn_on_channel feature is supported, the Host SHALL support the turn_on_channel feature parameters as defined in Table 9.15-26.

If the turn_on_channel feature is supported, the Card SHALL support the turn_on_channel feature parameters as defined in Table 9.15-26.

Table 9.15-26 - Turn-on Virtual Channel

Syntax	No. of Bits	Mnemonic
turn_on_channel() {		
reserved	3	0x0
turn_on_channel_defined	1	bslbf
if (turn_on_channel_defined == 1) {		
turn_on_virtual_channel	12	uimsbf
}		
else {		
reserved	12	uimsbf
}		
}		

turn_on_channel_defined

0b turn_on_channel not defined.
1b turn_on_channel defined.

turn_on_virtual_channel Virtual channel to be tuned to by the Host upon power up.

9.15.5.7.15 Terminal Association

This generic feature allows the Business System and the Conditional Access Controller to identify the Card and the Host.

When operating in M-Mode, the Host SHALL support the terminal_association feature as defined in Table 9.15-27.

When operating in M-Mode, the Card MAY support the terminal_association feature as defined in Table 9.15-27.

When operating in S-Mode, the Host MAY support the terminal_association feature as defined in Table 9.15-27.

When operating in S-Mode, the Card MAY support the terminal_association feature as defined in Table 9.15-27.

Table 9.15-27 - Terminal_Association

Syntax	No. of Bits	Mnemonic
terminal_association (){		
identifier_length	16	uimsbf
for (i=0; i<identifier_length; i++){		
terminal_assoc_identifier	8	uimsbf
}		
}		

identifier_length The number of bytes to follow

terminal_assoc_identifier 7-bit ASCII alphanumeric value assigned to the Host. The value may be utilized by the Host to associate itself with other Host devices residing on a local area network (e.g., a home network).

9.15.5.7.16 Common Download Group ID Assignment

This generic feature allows for a Group ID assignment for Host(s) when Common Download [CDL] is utilized. The vendor_id and hardware_version_id values for the Host are the same as defined in [CDL].

When operating in M-Mode, the Host SHALL support the cdl_group_id generic feature as defined in Table 9.15-28.

When operating in M-Mode, the Card MAY support the cdl_group_id generic feature as defined in Table 9.15-28.

When operating in S-Mode, the Host MAY support the cdl_group_id generic feature as defined in Table 9.15-28.

When operating in S-Mode, the Card MAY support the cdl_group_id generic feature as defined in Table 9.15-28.

The Host SHALL store the group_id value in persistent memory after receiving the cdl_group_id generic feature from the Card.

- The Host SHALL store a default group_id value of 0x0000 in persistent memory in the absence of a cdl_group_id generic feature received from the Card.
- The Host SHALL set the group_id value to the default value when the Card is removed or a different Card is inserted while the Host is powered.

Table 9.15-28 - Common Download Group ID Assignment

Syntax	No. of Bits	Mnemonic
cdl_group_id() {		
group_id	16	uimsbf
}		

group_id Group ID value to be stored in the Host's persistent memory for use as defined in [CDL].

9.15.5.7.17 zip_code

This generic feature allows the Business System and the Conditional Access Controller to identify the Card and the Host using the existing postal zip code.

When operating in M-Mode, the Host SHALL support the Zip Code feature as defined in Table 9.15-29.

When operating in M-Mode, the Card MAY support the Zip Code feature as defined in Table 9.15-29.

When operating in S-Mode, the Host MAY support the Zip Code feature as defined in Table 9.15-29.

When operating in S-Mode, the Card MAY support the Zip Code feature as defined in Table 9.15-29.

Table 9.15-29 - Zip_Code

Syntax	No. of Bits	Mnemonic
zip_code (){		
zip_code_length	16	uimsbf
for (i=0; i<zip_code_length; i++){		
zip_code_byte	8	uimsbf
}		
}		

zip_code_length The number of zip_code_bytes to follow.

zip_code_byte 7-bit ASCII alphanumeric value that defines a 5 character or a 5+4 character ZIP code in the format of #####-####.

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.

The Card may request that the Host perform a diagnostic and report the status/result in response to a headend OOB message, or an SNMP message request to perform a diagnostic that is supported exclusively on the Host.

For M-Mode, this resource has been modified from Type 1. Type 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 Type 1 of this APDU, the information returned will only be for the primary transport stream.

The Host SHALL support the Generic Diagnostic Resource using the resource identifier(s) as defined in Table 9.16-1.

The Card SHALL open only one session to the Generic Diagnostic Control Resource using the resource identifier(s) as defined in Table 9.16-1.

Table 9.16-1 - Generic Diagnostic Support Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic Diagnostic Support	S-Mode	260	1	2	0x01040042
Generic Diagnostic Support	M-Mode	260	2	1	0x01040081
Generic Diagnostic Support	M-Mode	260	2	2	0x01040082

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 Host SHALL use the diagnostic IDs as defined in Table 9.16-3 in the ***diagnostic_cnf()*** APDU.

The Card SHALL use the diagnostic IDs as defined in Table 9.16-3 in the ***diagnostic_req()*** APDU.

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
Home Networking Status	0x0D
Host Information	0x0E
Reserved	0x0F-0xFF

9.16.1 diagnostic_req APDU

The Card SHALL send the ***diagnostic_req()*** APDU as defined in Table 9.16-4 for S-Mode and Table 9.16-5 for M-Mode to request the Host to perform a specific set of diagnostic functions and report the result/status of the diagnostics.

Table 9.16-4 - S-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`
`  }`		
`}`		

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 1)

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. For parameters where this has no meaning, the value SHALL be 0x00.

9.16.2 diagnostic_cnf APDU

The Host SHALL send the ***diagnostic_cnf()*** APDU as defined in Table 9.16-6 for S-Mode and Table 9.16-7 for M-Mode after reception of the ***diagnostic_req()*** APDU and the Host has completed any tests required to report the result/status. When there are multiple instances of a diagnostic report, the Host SHALL send each report with the same diagnostic ID in the ***diagnostic_cnf()*** APDU. In this case, the number_of_diag value will be different than the one in the ***diagnostic_req()*** APDU.

Table 9.16-6 - S-Mode - diagnostic_cnf APDU Syntax (Type 1, Version 2)

Syntax	No. of Bits	Mnemonic
diagnostic_cnf() {		
diagnostic_cnf_tag	24	uimsbf
length_field()		
number_of_diag	8	uimsbf
for (i=o; 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()		
}		
if (diagnostic_id == 0x0D) {		
home_network_report()		
}		
if (diagnostic_id == 0x0E) {		
host_information_report()		
}		
}		
}		
}		

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.

status_field Status of the requested diagnostic. See Table 9.16-8.

Table 9.16-7 - M-Mode - diagnostic_cnf APDU Syntax (Type 2, Version 1)

Syntax	No. of Bits	Mnemonic
diagnostic_cnf() {		
diagnostic_cnf_tag	24	uimsbf
length_field()		
number_of_diag	8	uimsbf
for (i=o; i<number_of_diag; i++) {		
diagnostic_id	8	uimsbf
ltsid	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()		
}		
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()		
}		
if (diagnostic_id == 0x0D) {		
home_network_report()		
}		

Syntax	No. of Bits	Mnemonic

`        if (diagnostic_id == 0x0E) {`
`           host_information_report ()`
`        }`
`     }`
`  }`
`}`		

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. For parameters where this has no meaning, the value will be 0x00.

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

9.16.3 Diagnostic Report Definition

Each applicable diagnostic consists of a set of diagnostic reports that 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

In response to a memory_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with a memory_report as defined in Table 9.16-9 in the ***diagnostic_cnf()*** APDU.

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 Internal Hard drive, no DRM (Digital Rights Management) support
0x06 Video memory
0x07 Other memory
0x08 Internal Hard Drive, DRM support
0x09 External Hard Drive, no DRM support
0x0A External Hard Drive, DRM support
0x0B Optical media, no DRM support
0x0C Optical media, DRM support
0x0D-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

In response to a software_ver_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with a software_ver_report as defined in Table 9.16-10 for S-Mode and Table 9.16-11 for M-Mode in the ***diagnostic_cnf()*** APDU.

Table 9.16-10 - S-Mode 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.

Table 9.16-11 - M-Mode 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_length	8	uimsbf
for (j=0; j<application_version_length; j++) {		
application_version_byte	8	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_length Designates the number of characters required to define the application version

application_version_byte ASCII character, 8-bits per character, a string that identifies the application version.

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

In response to a firmware_ver_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with the firmware_ver_report as defined in Table 9.16-12 for S-Mode and Table 9.16-13 for M-Mode in the ***diagnostic_cnf()*** APDU.

Table 9.16-12 - S-Mode 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.

Table 9.16-13 - M-Mode firmware_ver_report

Syntax	No. of Bits	Mnemonic
firmware_ver_report() {		
firmware_version_length	8	uimsbf
for (j=0; j<firmware_version_length; j++) {		
firmware_version_byte	8	uimsbf
}		
firmware_date{		
firmware_year	16	uimsbf
firmware_month	8	uimsbf
firmware_day	8	uimsbf
}		
}		

firmware_version_length Designates the number of characters required to define the firmware version

firmware_version_byte ASCII character, 8-bits per character, a string that identifies the firmware version.

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

In response to a MAC_address_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with the MAC_address_report as defined in Table 9.16-14 in the ***diagnostic_cnf()*** APDU.

Table 9.16-14 - MAC_address_report

Syntax	No. of Bits	Mnemonic
MAC_address_report() {		
number_of_addresses	8	uimsbf
for (i=0; i<number_of_addresses; i++) {		
MAC_address_type	8	uimsbf
number_of_bytes	8	uimsbf
for (j=0; j<number_of_bytes; j++) {		
MAC_address_byte	8	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 in the ***diagnostic_req()*** APDU, the Host SHALL reply with a FAT_status_report as defined in Table 9.16-15 in the ***diagnostic_cnf()*** APDU. If a Host contains multiple FAT tuners, then the Host SHALL provide multiple FAT_status_reports in the ***diagnostic_cnf()*** APDU, one for each tuner.

Table 9.16-15 - FAT_status_report

Syntax	No. of Bits	Mnemonic
FAT_status_report() {		
reserved	4	'1111'
PCR_lock	1	bslbf
modulation_mode	2	bslbf
carrier_lock_status	1	bslbf
SNR	16	tcimsbf
signal_level	16	tcimsbf
}		

PCR_lock Indicates if the FAT channel receiver is locked to the currently tuned channel. (**Note:** Not valid if modulation_mode == 00b OR modulation_mode == 0b11)

0b Not locked
1b Locked

modulation_mode Indicates if the current forward transport is analog, QAM-64, or QAM-256

00b Analog

01b QAM64
10b QAM256
11b Other

carrier_lock_status Indicates if the current carrier is locked or not locked. (**Note:** Not valid if modulation_mode == 00b OR modulation_mode == 0b11)

0b Not locked
1b Locked

SNR Numerical representation of the signal to noise ratio in tenths of a dB. (**Note:** Not valid if modulation_mode == 00b OR modulation_mode == 0b11)

signal_level Numerical representation of the signal level in tenths of a dBmV. **Note:** For modulation_mode == 00b, use peak signal level. All others use average signal level.

9.16.3.6 FDC_status_report

In response to an FDC_status_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with an FDC_status_report as defined in Table 9.16-16 in the ***diagnostic_cnf()*** APDU.

Table 9.16-16 - FDC_status_report

Syntax	No. of Bits	Mnemonic
FDC_report() {		
FDC_center_freq	16	uimsbf
reserved	6	'111111'
carrier_lock_status	1	bslbf
reserved	1	bslbf
}		

FDC_center_freq Indicates the frequency of the FDC center frequency, in MHz. (Frequency = value * 0.05 + 50 MHz).

Table 9.16-17 - 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)							

carrier_lock_status Indicates if the current carrier is locked or not locked.

0b Not locked
1b Locked

9.16.3.7 current_channel_report

In response to a current_channel_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with a current_channel report as defined in Table 9.16-18 in the ***diagnostic_cnf()*** APDU. If a Host contains multiple FAT tuners, then the Host SHALL send multiple current_channel reports in the ***diagnostic_cnf()*** APDU, one for each tuner.

Table 9.16-18 - current_channel_report

Syntax	No. of Bits	Mnemonic
current_channel_() {		
Reserved	2	'11'
channel_type	1	bslbf
authorization_flag	1	bslbf
purchasable_flag	1	bslbf
purchased_flag	1	bslbf
preview_flag	1	bslbf
parental_control_flag	1	bslbf
current_channel	16	uimsbf
}		

channel_type Indicates if the channel is analog or digital.

0b Analog
1b Digital

authorization_flag Indicates if the Host is authorized for the currently tuned channel.

0b Not authorized
1b Authorized

purchasable_flag Indicates if the currently tuned channel may be purchased.

0b Not purchasable
1b Purchasable

purchased_flag Indicates if the currently tuned channel has been purchased.

0b Not purchased
1b Purchased

preview_flag Indicates if the currently tuned channel is in preview mode.

0b Not in preview mode
1b In preview mode

parental_control_flag Indicates if the currently tuned channel is under parental control.

0b Channel is not blocked
1b Channel is blocked

current_channel Indicates the numerical representation of the currently tuned channel.

If a tuner is not being utilized, then the Host SHALL return 0xFFFF for the current_channel parameter of the current_channel_report in the ***diagnostic_cnf()*** APDU.

9.16.3.8 1394_port_report

In response to a 1394_port_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with a 1394_port_report as defined in Table 9.16-19 for S-Mode, and Table 9.16-20 for M-Mode in the ***diagnostic_cnf()*** APDU.

Table 9.16-19 - S-Mode 1394_port_report

Syntax	No. of Bits	Mnemonic
1394_port_report() {		
reserved	3	'111'
loop_status	1	bslbf
root_status	1	bslbf
cycle_master_status	1	bslbf
port_1_connection_status	1	bslbf
port_2_connection_status	1	bslbf
total_number_of_nodes	16	uimsbf
}		

Table 9.16-20 - M-Mode 1394_port_report

Syntax	No. of Bits	Mnemonic
1394_port_report() {		
reserved	2	'11'
loop_status	1	bslbf
root_status	1	bslbf
cycle_master_status	1	bslbf
host_a/d_source_selection_status	1	bslbf
port_1_connection_status	1	bslbf
port_2_connection_status	1	bslbf
total_number_of_nodes	16	uimsbf
number_of_connected_devices	8	uimsbf
for (i=0;		
i<number_of_connected_devices; i++) {		
device_subunit_type	5	uimsbf
device_a/d_source_selection_status	1	bslbf
reserved	2	'11'
eui_64	64	uimsbf
}		
}		

loop_status Indicates if a loop exists on the 1394 bus.

- 0b No loop exists
- 1b Loop exists

root_status Indicates if the Host device is the root node on the 1394 bus.

- 0b Not root
- 1b Is root

cycle_master_status Indicates if the Host device is the cycle master node on the 1394 bus.

- 0b Not cycle master
- 1b Is cycle master

host_device_a/d_source_selection_status Indicates if the Host supports A/D source selection function.

- 0b Not Supported
- 1b Supported

port_1_connection_status Indicates if port 1 of the 1394 PHY is connected to a 1394 bus.

- 0b Not connected
- 1b Connected

port_2_connection_status Indicates if port 2 of the 1394 PHY is connected to a 1394 bus.

- 0b Not connected
- 1b Connected

total_number_of_nodes Indicates the total number of nodes connected to the 1394 bus.

number_of_connected_devices Total number of sink devices connected to the Host via IEEE-1394.

device_subunit_type Subunit type of device connected to the Host, where subunit type encodes are as defined via the 1394TA:

0x00 Monitor
0x01 Audio
0x02 Printer
0x03 Disc
0x04 Tape Recorder/Player
0x05 Tuner
0x06 CA
0x07 Camera
0x08 Reserved
0x09 Panel
0x0A Bulletin Board
0x0B Camera Storage
0x0C-0x1B Reserved
0x1C Vendor Unique
0x1D Reserved
0x1E Subunit_type extended to next byte
0x1F Unit

device_a/d_source_selection_status Indicates if the device supports A/D source selection function.

0b Not Supported
1b Supported

eui_64 64-bit Extended Unique Identifier (a.k.a. Global Identifier) of the device**.**

9.16.3.9 DVI Status Report

In response to a DVI_status_report request in the ***diagnostic_req()*** APDU, if the Host supports the DVI interface, the Host SHALL reply with a DVI_status_report as defined in Table 9.16-21 in the ***diagnostic_cnf()*** APDU. A Host SHALL provide a DVI_status_report if it has a DVI connector, even if an HDMI device is connected through the DVI connector. If a Host does not have a DVI connector, the Host SHALL NOT respond to the DVI Status Report Request, even if a DVI device is connected through an HDMI connector.

Table 9.16-21 - DVI Status Report Syntax

Syntax	No. of bits	Mnemonic
DVI_status_report() {		
reserved	3	'111'
connection_status	2	bslbf
host_HDCP_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 DVI port

- 00b No connection exists
- 01b Device connected - not repeater
- 10b Device connected - repeater
- 11b Reserved

host_HDCP_status Indicates if HDCP is enabled on the DVI 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 $00_{2)}$.

- 00b Non HDCP device
- 01b Compliant HDCP device
- 10b Revoked HDCP device
- 11b Reserved

video_format Indicates the current video format utilized on the DVI port as defined in the following fields:

horizontal_lines Indicates the number of horizontal lines associated with the video format on the DVI link.

vertical_lines Indicates the number of vertical lines associated with the video format on the DVI link.

frame_rate Indicates the frame rate associated with the video format on the DVI link as defined in the following table.

Table 9.16-22 - 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

aspect_ratio Indicates the aspect ratio associated with the video format on the DVI link as defined in the following table.

Table 9.16-23 - 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

prog_inter_type 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 eCM_status_report request in a ***diagnostic_req()*** APDU, the Host SHALL reply with an eCM_status_report as defined in Table 9.16-24 in the ***diagnostic_cnf()*** APDU.

Table 9.16-24 - eCMStatus Report Syntax

Syntax	No. of bits	Mnemonic
eCM_status_report() {		
downstream_center_freq	16	uimsbf
downstream_power_level	16	tcimsbf
downstream_carrier_lock_status	1	bslbf
reserved	2	"11"
channel_s-cdma_status	2	bslbf
upstream_modulation_type	3	bslbf
upstream_xmt_center_freq	16	uimsbf
upstream_power_level	16	tcimsbf
upstream_symbol_rate	8	uimsbf
}		

downstream_center_freq Indicates the frequency of the FDC center frequency, in MHz (Frequency = value * 0.05 + 50 MHz).

Table 9.16-25 - 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.

downstream carrier_lock_status Indicates if the current carrier is locked or not locked.

0b Not locked
1b Locked

channel_s-cdma_status Channel S-CDMA status

00b Channel is not S-CDMA
01b Channel is S-CDMA, TCM encoding
10b Channel is S-CDMA, TDMA encoding
11b Channel is S-CDMA, other encoding

upstream_modulation_type Indicates the current upstream modulation type.

000b QPSK
001b 16-QAM
010b 32-QAM
011b 64-QAM
100b 128-QAM
101b 256-QAM
110b 512-QAM
111b Other

upstream_xmt__center_freq Indicates the frequency of the RDC center frequency, in MHz (Frequency = value * 0.05 + 5 MHz).

Table 9.16-26 - 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 5/12 Msps
0x06-0xFF Reserved

upstream_modulation_type Indicates the current modulation type.

0b QPSK
1b 16QAM

9.16.3.11 HDMI Port Status Report

In response to an HDMI_status_report request in a ***diagnostic_req()*** APDU, if the Host supports the HDMI interface, the Host SHALL reply with an HDMI_status_report as defined in Table 9.16-27 in the ***diagnostic_cnf()*** APDU. A Host SHALL provide an HDMI_status_report if it has an HDMI connector, even if a DVI device is connected through the HDMI connector. If a Host does not have an HDMI connector, the Host SHALL NOT respond to the HDMI Status Report Request, even if an HDMI device is connected through a DVI connector.

Table 9.16-27 - HDMI Status Report Syntax

Syntax	No. of bits	Mnemonic
HDMI_status_report() {		
device_type	1	bslbf
color_space	2	bslbf
connection_status	2	bslbf
host_HDCP_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
}		
audio_format		
{		
audio_sample_size	2	bslbf
audio_format	3	bslbf
audio_sample_freq	3	bslbf
}		
}		

device_type Indicates whether the device is DVI or HDM

0b Device connected through HDMI connector uses DVI

1b Device connected through HDMI connector uses HDMI

color_space Indicates the color space utilized (valid when connection_status does not equal 0 AND device_type is equal to 0b1)

00b RGB
01b YCC422
10b YCC444
11b Reserved

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_2)

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 9.16-28 - Frame Rate Associated With the Video Format On the HDMI 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

aspect_ratio Indicates the aspect ratio associated with the video format on the HDMI link as defined in the following table:

Table 9.16-29 - Aspect Ratio Associated With the Video Format On the HDMI Link

Bit Value	Video Format
00	4:3

Bit Value	Video Format
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

audio_sample_size Audio sample size (valid when connection_status is not equal to 0 AND device_type is equal to 0b1 and audio_format = 0b000)

00b Not valid (audio_format is not equal to 0b000)
01b 16
10b 20
11b 24

audio_format Audio format (valid when connection_status is not equal to 0 AND device_type is equal to 0b1)

000b PCM
001b MPEG-1
010b MPEG-2
011b DTS
100b AAC
101b MP3
110b ATRAC
111b Other audio format

audio_sample_freq Audio sample frequency (valid when connection_status is not equal to 0 AND device_type is equal to 0b1)

000b 32.0 KHz
001b 44.1 KHz
010b 48.0 KHz
011b 88.2 KHz
100b 96.0 KHz
101b 176.4 KHz
110b 192 KHz
111b Other sample frequency

9.16.3.12 RDC Status Report

In response to an RDC_status_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with an RDC_status_report as defined in Table 9.16-30 in the ***diagnostic_cnf()*** APDU.

Table 9.16-30 - RDC_status_report

Syntax	No. of Bits	Mnemonic

RDC_report() {
  RDC_center_freq
  RDC_transmitter_power_level
  reserved
  RDC_data_rate
}
	
	16
	8
	6
	2
		
		uimsbf
		tcimsbf
		'111111'
		bslbf

RDC_center_freq Indicates the frequency of the RDC center frequency, in MHz (Frequency = value * 0.05 + 5 MHz).

Table 9.16-31 - 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

In response to a net_address_report request in a ***diagnostic_req()*** APDU, the Host SHALL respond with a net_address_report as defined in Table 9.16-32 in the ***diagnostic_cnf()*** APDU.

Table 9.16-32 - net_address_report

Syntax	No. of Bits	Mnemonic
net_address_report() {		
number_of_addresses	8	uimsbf
for (i=0; i<number_of_addresses; i++) {		
net_address_type	8	uimsbf
number_of_bytes_net	8	uimsbf
for (j=0; j<number_of_bytes_net; j++) {		
net_address_byte	8	uimsbf
}		
number_of_bytes_subnet	8	uimsbf
for (j=0; j<number_of_bytes_subnet; j++) {		
sub_net_address_byte	8	uimsbf
}		
}		
}		

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_net The total number of bytes required for the network address. **Note:** IPv6 will be reported as 16 bytes.

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.

number_of_bytes_subnet The total number of bytes required for the subnet address. **Note:** IPv6 will be reported as 16 bytes.

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.

9.16.3.14 home_network_report

In response to a home_network_report request in a ***diagnostic_req()*** APDU, if the Host supports the home_network_report, the Host SHALL reply with a home_network_report as defined in Table 9.16-33 in the ***diagnostic_cnf()*** APDU.

Table 9.16-33 - home_network_report

Syntax	No. of Bits	Mnemonic
home_network_report() {		
max_clients	8	uimsbf
host_DRM_status	8	uimsbf
connected_clients	8	uimsbf
for (i=0; i<connected_clients; i++) {		
client_mac_address	48	uimsbf
number_of_bytes_net	8	uimsbf
for(j=0; j<number_of_bytes_net; j++) {		
client_IP_address_byte	8	uimsbf
}		
client_DRM_status	8	uimsbf
}		
}		

max_clients Maximum number of clients the Host can support.

If the Host does not support home network clients, then it SHALL report 0x00 for max_clients in a diagnostic home_network_report.

host_DRM_status Host DRM (Digital Rights Management) capability.

0x00 Host has no DRM capability.
0x01 Host supports DRM but not for home networked clients.
0x02 Host supports DRM for itself and home networked clients.
0x03-0xFF - Reserved.

connected_clients Number of connected clients.

client_mac_address MAC address of client i.

number-of_bytes_net Number of bytes in the network address. **Note:** IPv6 will be reported as 16 bytes.

client_IP_address_byte IP address of client i. **Note:** If no IP address is assigned for client i, then this value will be returned as 0x00 for all bytes.

client_DRM_status ASD status of client i.

0x00 - No DRM support in client i.
0x01 - DRM trust not established in client i.
0x02 - DRM trust established in client i.host_information_report
0x03 - 0xFF - Reserved

9.16.3.15 host_information_report

In response to a host_information_report request in the ***diagnostic_req()*** APDU, the Host SHALL reply with a host_information_report in the ***diagnostic_cnf()*** APDU as defined in Table 9.16-34.

Table 9.16-34 - host_information_report

Syntax	No. of Bits	Mnemonic
host_information_report() {		
vendor_name_length	8	uimsbf
for (i=0; i<vendor_name_length; i++) {		
vendor_name_character	8	uimsbf
}		
model_name_length	8	uimsbf
for (i=0; model_name_length; I++) {		
model_name_character	8	uimsbf
}		
if(type==2 && version==2){		
vendor_id	24	uimsbf
hardware_id	32	uimsbf
group_id	16	uimsbf
}		
}		

vendor_name_length Length of the vendor name.

vendor_name_character Name of the vendor in ASCII.

model_name_length Length of the model name.

model_name_character Name of the model in ASCII.

vendor_id Vendor ID assigned to Host as defined in [CDL].

hardware_id Hardware ID assigned to Host as defined in [CDL].

group_id Group ID assigned to Host as defined in [CDL]. **Note:** If no group ID has been assigned, then a value of 0x0000 is used.

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. 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 more than one Specific Application Support (SAS) session 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, an ***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).

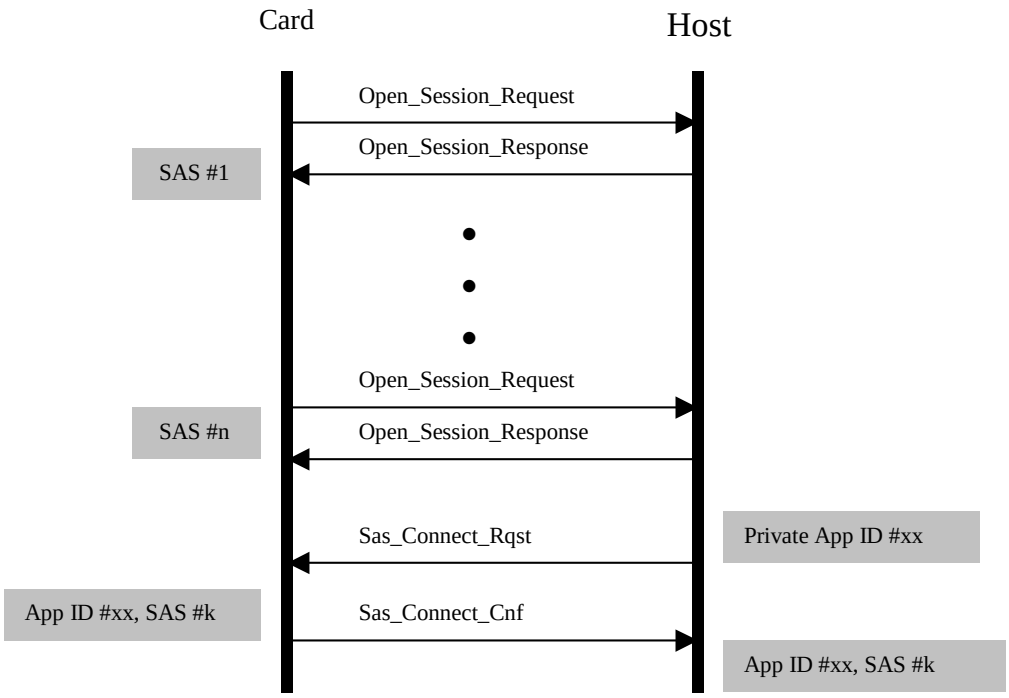


Figure 9.17-1 - Specific Application Support Connection Sequence

In some instances, the Card may receive an ***SAS_connect_rqst()*** APDU before a session has been opened for the associated vendor-specific application.

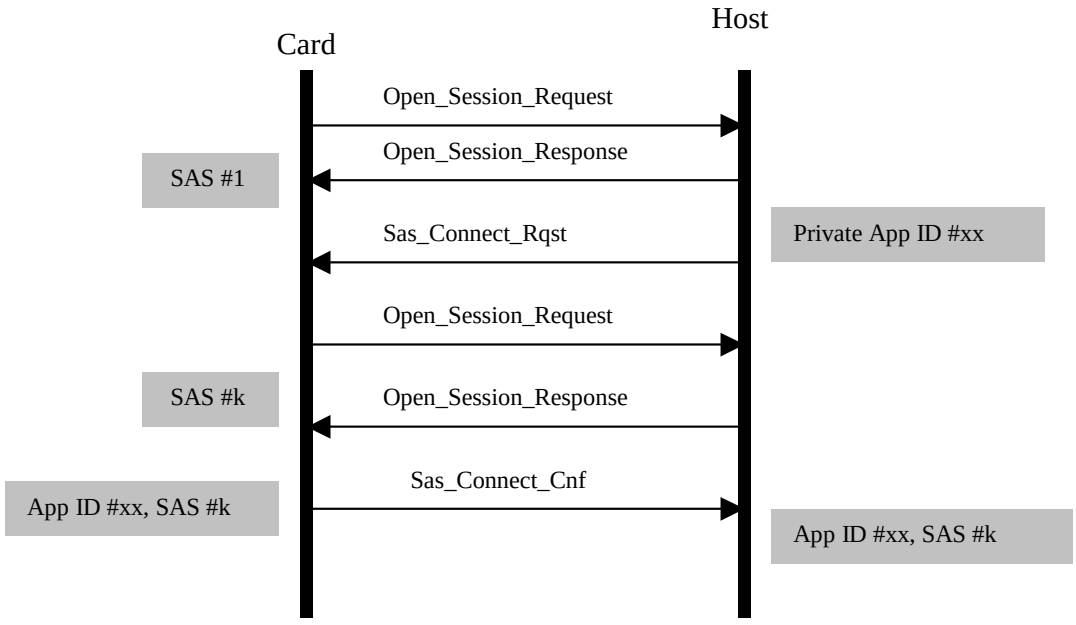


Figure 9.17-2 - Specific Application Support Alternate Connection Sequence

The Host SHALL support the Specific Application Support (SAS) Resource using the resource identifier as defined in Table 9.17-1.

The Host SHALL support up to 32 sessions to the SAS resource.

The Card SHALL open at least one session to the SAS resource on the Host, using the resource identifier as defined in Table 9.17-1.

If there is only one SAS resource session open, the Card SHALL NOT close the open session to the SAS resource.

If there is only one SAS resource session open, the Host SHALL NOT close the open session to the SAS resource.

Note: After a PCMCIA or Card Reset, all sessions need to be reopened by the Card, including SAS sessions. A PCMCIA or Card Reset implicitly closes all sessions, resetting the session count to zero (0).

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 eight 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_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	↔

9.17.1 SAS_connect_rqst APDU

The Host SHALL send an ***SAS_connect_rqst()*** APDU as defined in Table 9.17-3 to the Card to establish a connection between a private Host 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 from the Host, the Card SHALL reply with an ***SAS_connect_cnf()*** APDU as defined in Table 9.17-4 to inform the Host of which SAS session is to be used for this connection.

After receiving the ***SAS_connect_rqst()*** APDU from the Host, if a session is not available to the SAS Resource for the application, the Card SHALL open a new session to the SAS Resource prior to responding with the ***SAS_connect_cnf()*** APDU.

Table 9.17-4 - SAS_connect_cnf APDU Syntax

Syntax	No. of Bits	Mnemonic
SAS_connect_cnf() {		
SAS_connect_cnf_tag	24	uimsbf
length_field()		
private_host_application_ID	64	uimsbf
SAS_session_status	8	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. A receipt of the ***SAS_data_rqst()*** APDU is an indication 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_rqst()*** for the life of a connection, it may send more than one ***SAS_data_rqst()*** APDU over the lifetime of an SAS connection.

After an SAS connection has been established, the Host MAY send the ***SAS_data_rqst()*** APDU as defined in Table 9.17-5 to indicate to an application on the Card that the application on the Host is ready to process incoming data.

After an SAS connection has been established, the Card MAY send the ***SAS_data_rqst()*** APDU as defined in Table 9.17-5 to indicate to an application on the Host that the application on the Card is ready to process incoming data.

Table 9.17-5 - SAS_data_rqst APDU Syntax

Syntax	No. of Bits	Mnemonic

SAS_data_rqst() {
SAS_data_rqst_tag
length_field()
}
	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 an 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. A Host application MAY send an ***SAS_data_av()*** APDU as defined in Table 9.17-6 to indicate that the Host application has data to send across the CHI, only after an ***SAS_data_rqst()*** APDU has been received.

A Card application MAY send an ***SAS_data_av()*** APDU as defined in Table 9.17-6 to indicate that the Card application has data to send across the CHI, only after an ***SAS_data_rqst()*** APDU has been received.

Note: The data itself is sent in the ***SAS_query()*** and ***SAS_reply()*** APDUs.

Table 9.17-6 - SAS_data_av APDU Syntax

Syntax	No. of Bits	Mnemonic

SAS_data_av() {
SAS_data_av_tag
length_field()
SAS_data_status
transaction_nb
}
	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 is set to 0x00.

9.17.5 SAS_data_cnf APDU

After the Host application receives an ***SAS_data_av()*** APDU, it SHALL transmit the ***SAS_data_cnf()*** APDU as defined in Table 9.17-7 to acknowledge that it is preparing to receive the available data.

After the Card application receives an ***SAS_data_av()*** APDU, it SHALL send the ***SAS_data_cnf()*** APDU as defined in Table 9.17-7 to acknowledge that it is preparing to receive the available data, even if there is no data available.

Table 9.17-7 - SAS_data_cnf APDU Syntax

Syntax	No. of Bits	Mnemonic

SAS_data_av_cnf() {
SAS_data_av_cnf_tag
length_field()
transaction_nb
}
	24
	
	8
		uimsbf
		
		uimsbf

SAS_data_av_cnf_tag 0x9F9A04

transaction_nb The transaction_nb assigned in the ***SAS_data_av()*** APDU.

9.17.6 SAS_server_query APDU

After sending the ***SAS_data_cnf()*** APDU to the Card application, the Host application SHALL send an ***SAS_server_query()*** APDU as defined in Table 9.17-8 to request the transfer of application specific data from the Card application.

After sending the ***SAS_data_cnf()*** APDU to the Host application, the Card application SHALL send an ***SAS_server_query()*** APDU as defined in Table 9.17-8 to request the transfer of application specific data from the Host application.

Table 9.17-8 - SAS_server_query APDU Syntax

Syntax	No. of Bits	Mnemonic

SAS_server_query () {
SAS_server_query_tag
length_field()
transaction_nb
}
	24
	
	8
		uimsbf
		
		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 Host application SHALL respond with the ***SAS_server_reply()*** APDU as defined in Table 9.17-9 with the data to transfer.

After receiving the ***SAS_server_query()*** APDU, the Card application SHALL respond with the ***SAS_server_reply()*** APDU as defined in Table 9.17-9 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 utilized 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.

After an SAS connection has been established, the Host MAY send the ***SAS_async_msg()*** APDU as defined in Table 9.17-10 to send a message to a vendor-specific application on the Card.

After an SAS connection has been established, the Card MAY send the ***SAS_async_msg()*** APDU as defined in Table 9.17-10 to send a message to a private application on the Host.

Table 9.17-10 - SAS_Async Message APDU Syntax

Syntax	No. of Bits	Mnemonic
SAS_async_msg() {		
SAS_async_msg_tag	24	uimsbf
length_field()		
message_nb	8	uimsbf
message_length	16	uimsbf
for (i =0; i< message_length; i++)		
{		
message_byte	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

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 that encapsulate the previous operations of the resources but add 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 "non-viewing" state).

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.

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

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

While the Card is performing its upgrade operation, its ability to support the normal Card interface may range from severely limited to fully operational. 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 the 5-second timeout is specified in the ***firmware_upgrade()*** APDU, the Card SHALL still respond to the transport layer polls with a 5-second timeout during a firmware upgrade. If the 5-second timeout is specified in the ***firmware_upgrade()*** APDU, and the Card fails to respond to a transport layer 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. During

a firmware upgrade to the Card, 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. During a firmware upgrade to the Card, the Host SHALL respond to any session or application layer operation initiated by the Card and perform all related 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 is specified in the ***firmware_upgrade()*** APDU, and the download_timeout period expires, the Host SHALL perform a PCMCIA reset on the Card.
5. 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 can send the ***firmware_upgrade_complete()*** APDU with the reset type set to 0x02, "no reset required".

9.18.3.3 Flowchart (Informative)

Figure 9.18-1 is a flowchart showing the Card/Host interface, which uses Card upgrade methods defined in this document.

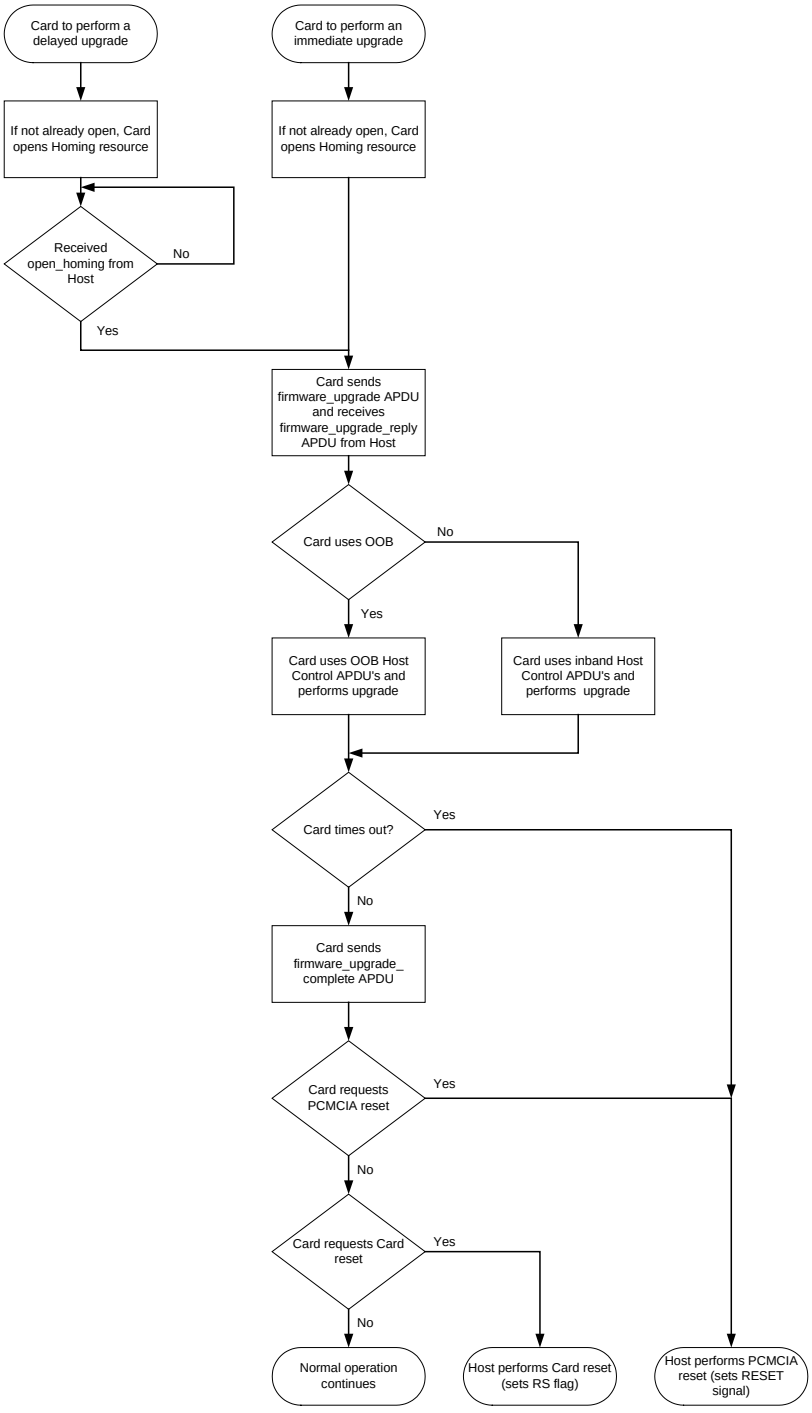


Figure 9.18-1 - Firmware Upgrade Flowchart

9.18.4 Homing Resource

9.18.4.1 Homing Resource Definition

The Homing resource allows for the Card to upgrade its firmware using Host resources.

Table 9.18-1 - Homing Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Homing	S-Mode/M-Mode	17	1	2	0x00110042

The Host SHALL support the Homing Resource using the resource identifier as specified in Table 9.18-1.

The Card SHALL open a maximum of one session to the Homing resource using the resource identifier as specified in Table 9.18-1 when it requires a firmware upgrade.

The Homing resource includes the following APDUs:

Table 9.18-2 - Homing APDUs

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 Host SHALL send the ***open_homing()*** APDU as defined in Table 9.18-3 to the Card when it enters the standby state, either from power up or from user action, independent of whether the Host Control resource has a session active.

Table 9.18-3 - Open Homing Object Syntax

Syntax	No. of bits	Mnemonic

`open_homing() {`
`  open_homing_tag`
`  length_field()`
`}`
	24	uimsbf

open_homing_tag 0x9F9990

9.18.4.3 open_homing_reply ()

The Card SHALL respond with the ***open_homing_reply()*** APDU as defined in Table 9.18-4 to acknowledge receipt of the ***open_homing()*** APDU.

Table 9.18-4 - Open Homing Reply Object Syntax

Syntax	No. of bits	Mnemonic

open_homing_reply() {
  open_homing_reply_tag
  length_field()
}
	24	uimsbf

open_homing_reply tag 0x9F9992

9.18.4.4 homing_active

After receipt of the ***open_homing_reply()*** APDU, the Host SHALL send the ***homing_active()*** APDU as defined in Table 9.18-5 to inform the Card that the homing request has been activated.

Table 9.18-5 - Homing Active Object Syntax

Syntax	No. of bits	Mnemonic

homing_active() {
  homing_active_tag
  length_field()
}
	24	uimsbf

homing_active_tag 0x9F9993

9.18.4.5 homing_cancelled

The Host SHALL close the homing state by sending a ***homing_cancelled()*** APDU as defined in Table 9.18-6 to the Card.

Table 9.18-6 - Homing Cancelled Object Syntax

Syntax	No. of bits	Mnemonic

homing_cancelled() {
  homing_cancelled_tag
  length_field()
}
	24	uimsbf

homing_canceled_tag 0x9F9991

9.18.4.6 homing_complete

When the Card no longer needs the homing function, it SHALL send a ***homing_complete()*** APDU as defined in Table 9.18-7 to the Host.

Table 9.18-7 - Homing Complete Object Syntax

Syntax	No. of bits	Mnemonic

homing_complete() {
  homing_complete_tag
  length_field()
}
	24	uimsbf

homing_complete_tag 0x9F9994

9.18.4.7 firmware_upgrade

The Card SHALL send the ***firmware_upgrade()*** APDU as defined in Table 9.18-8 to the Host to inform the Host that the Card is commencing a firmware upgrade. If the upgrade_source in the ***firmware_upgrade()*** APDU is equal to the QAM inband channel (01), the Host SHALL provide access to the inband tuner through the Host Control resource APDUs. The Host SHALL NOT interrupt a firmware upgrade to the Card until it receives the ***firmware_upgrade_complete()*** APDU from the Card. If the Host is not in the standby mode when a firmware upgrade is requested, the Host SHALL display the user_notification_text field in the ***firmware_upgrade()*** APDU formatted according to [ISO8859-1].

Table 9.18-8 - Firmware Upgrade Object Syntax

Syntax	No. 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 Defines 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 SHALL send the ***firmware_upgrade_reply()*** APDU as defined in Table 9.18-9 in response to the ***firmware_upgrade()*** APDU. The Card SHALL NOT start a firmware download operation until it receives the ***firmware_upgrade_reply()*** APDU.

Table 9.18-9 - Firmware Upgrade Reply Object Syntax

Syntax	No. of bits	Mnemonic

firmware_upgrade_reply() {
  firmware_upgrade_reply_tag
  length_field()
}
	24	uimsbf

firmware_upgrade_reply_tag 0x9F9996

9.18.4.9 firmware_upgrade_complete

After the Card has completed its upgrade or cancels the firmware upgrade, it SHALL send the ***firmware_upgrade_complete()*** APDU as defined in Table 9.18-10 to the Host. Included in this is whether the Card needs a PCMCIA reset (RESET signal active), Card reset (RS flag active) (applies in S-Mode only), 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	No. of bits	Mnemonic

Firmware_upgrade_complete() {
  firmware_upgrade_complete_tag
  length_field()
  reset_request_status
}
	24
	
	8
		uimsbf
		
		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 (S-Mode only)
- 0x02 No reset required - Normal Operation continues
- 0x03 0xFF Reserved

Note that if the Card cancels the firmware upgrade, it can send the ***firmware_upgrade_complete()*** APDU with no reset required. Normal operation should continue if the Host receives this APDU.

9.19 Support for Common Download

The Card SHALL support the common download protocol as defined in [CDL].

The [CDL] specification defines a protocol for Host devices to upgrade their operating software image.

9.20 DSG Resource

9.20.1 DSG Mode

In advanced_DSG_mode and advanced_DSG_one-way_mode, all SCTE 65 SI messages, SCTE 18 EAS messages, and CVTs and OCAP XAITs are received either directly by the Host or are received over the extended channel. The Host determines this based on the presence of DSG Broadcast Tunnel types defined in the Host Entries section of the ***DSG_directory()*** APDU. If the Host Entries section indicates a Broadcast Tunnel of a particular type, then the data is received directly by the Host via a DSG Broadcast Tunnel. If the Host Entries do not indicate a Broadcast Tunnel of a particular type, then the data may be delivered over the extended channel. As an example: the Host Entries

indicates the presence of a Broadcast Tunnel of type SCTE 18 (dsg_client_id = Broadcast Client ID for SCTE 18 = 0x01 0x02 0x00 0x02) and no other types, thus indicating that the Host must consume SCTE 18 EAS messages via the Broadcast Tunnel and request an extended channel MPEG flow for the SCTE 65, CVTs and OCAP XAIT messages.

The following messages are used for DSG configuration and operation:

- ***inquire_DSG_mode ()*** - The Host can inquire of the Card the preferred operational mode for the network.
- ***set_DSG_mode ()*** -The Card commands the Host to operate in the preferred operational mode for the network; either SCTE 55_mode, advanced_DSG_mode, or advanced_DSG_one-way_mode.
- ***send_DCD_info ()*** - The Host uses the ***send_DCD_info()*** message to pass the TLVs contained in the DCD message.
- ***DSG_directory ()*** - The Card uses the ***DSG_directory()*** message to pass DSG Advanced mode configuration parameters to the Host.
- ***DSG_message ()*** - This message is used by the Host to pass the upstream channel ID (UCID) to the Card or to indicate certain eCM operational states.
- ***DSG_error ()*** - The Card can inform the Host of errors that occur while operating in DSG mode.

The Host SHALL support the DSG Resource using the resource identifier as defined in Table 9.20-1.

The Host SHALL default to SCTE 55 mode until the ***set_DSG_mode()*** APDU is received.

The Card SHALL open a maximum of one session to the DSG Resource using the identifier as defined in Table 9.20-1.

Table 9.20-1 - DSG Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
DSG	S-Mode/M-Mode	4	1	1	0x00040041
DSG	S-Mode/M-Mode	4	1	2	0x00040042

NOTE: Type 1, Version 2 deprecates the use of DSG Basic mode.

The DSG resource as defined in Table 9.20-1 is optional for Card operating in S-Mode, mandatory for Card operating in M-Mode, and mandatory for Host operating in either S or M mode.

The DSG Resource APDU messages are as follows:

Table 9.20-2 - DSG APDUs

APDU Name	Tag Value	Resource	Direction Host ↔ Card	
			Host modem	Card modem
inquire_DSG_mode()	0x9F9100	DSG	→	→
set_DSG_mode()	0x9F9101	DSG	←	←
DSG_error()	0x9F9102	DSG	←	N/A
DSG_message()	0x9F9103	DSG	→	N/A
DSG_directory()	0x9F9104	DSG	←	N/A
send_DCD_info()	0x9F9105	DSG	→	N/A

9.20.1.1 DSG Advanced Mode

- The Host scans for a downstream DOCSIS channel containing a DCD message upon receipt of a ***set_dsg_mode ()*** APDU with an operational_mode value = 0x03 or 0x04.
- The Host passes the contents (i.e., TLVs) of the first DCD message received on a downstream channel (after reassembling any fragmentation) to the Card using the ***send_DCD_info ()*** APDU, regardless of the configuration count change field. After the initial ***send_DCD_info ()*** message has been sent, the Host only sends the DCD message when it detects a change in the configuration count change field in the DCD message, detects eCM MAC layer reinitialization, or after the completion of the DCC operation. The DCD message is defined in [DSG].
- To inform the Host that the DSG channel is not valid, the Card will use the DSG_error() APDU with error status = 0x01 - Invalid DSG Channel. The Host then searches for another DOCSIS Channel containing a DCD message. How the Card determines that a DSG channel is not valid is outside the scope of this specification.
- If the Card determines that the DSG channel is valid, then the Host will stay on the downstream and forward requested DSG data flows to the Card or terminate DSG data flows directly.
- The Card passes the DSG Configuration information received in the DCD message to the Host using the ***DSG_directory()*** APDU upon selection of a valid DSG channel or whenever the Card determines that it is necessary.
- The Host sends the ***DSG_message()*** to pass the UCID, when identified, to the Card. The Host sends the ***DSG_message()*** whenever it detects a change in the UCID value.
- The Card MAY use the Upstream Channel ID (UCID) passed by the Host in the ***DSG_message()*** to select appropriate DSG filters when UCIDs are specified in the DSG rules.
- After the Card parses the DCD message, the Card uses the ***DSG_directory()*** APDU to provide the Host with a set of MAC Addresses and DSG classifiers as applicable for specific DSG data flows.
- The Card MAY resend an updated ***DSG_directory()*** APDU at any time when operating in advanced DSG mode.
- Host specific DSG filters are indicated by the presence of the number_of_host_entries > 0 in the ***DSG_directory()*** APDU, where dir_entry_type = 0x01.
- DSG filters requested by the Card are defined in the number_of_card_entries loop in the ***DSG_directory()*** APDU. All DSG filters defined in the number_of_card_entries loop are forwarded to the eCM.
- The Host uses DSG classifiers provided to it in the Card section of the ***DSG_directory()*** APDU to filter DSG data packets for transmission to the Card.
- The Host uses DSG classifiers provided to it in the Host section of the ***DSG_directory()*** APDU to filter DSG data packets for DSG Clients on the Host.
- The ***DSG_directory()*** APDU may define identical filters in the Host Entries loop and Card Entries loop; in this case the Host consumes the DSG data packets directly in addition to sending these packets to the Card.

The following figure is an example of the initial message exchange between the Card and the Host for Advanced Mode Operation:

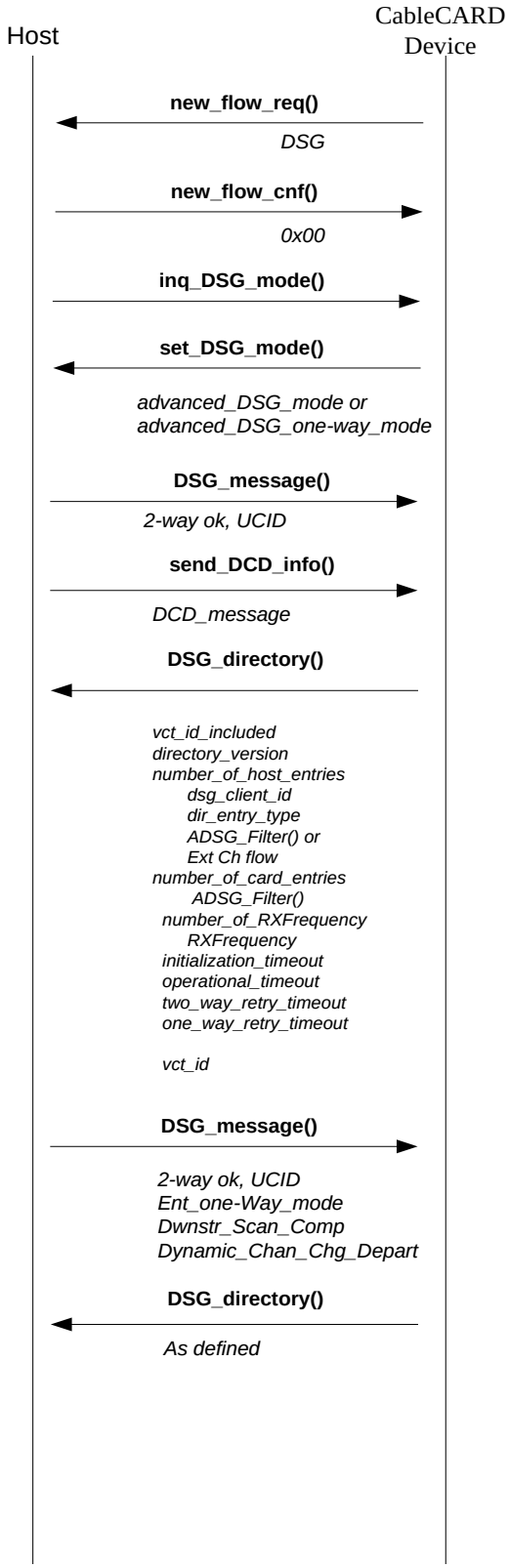


Figure 9.20-1 - Sample Advanced Mode Message Flow

9.20.2 inquire_DSG_mode APDU

The Host SHALL send the ***inquire_DSG_mode ()*** APDU as defined in Table 9.20-3 to determine the preferred operational mode for the network, either QSPK mode or DSG mode.

Table 9.20-3 - inquire_DSG_mode APDU Syntax

Syntax	No. of Bits	Mnemonic

inquire_DSG_mode() {
  inquire_DSG_mode_tag
  length_field() /* always = 0x00 */
}
	24	uimsbf

inquire_DSG_mode_tag 0x9F9100

9.20.3 set_DSG_mode APDU

Upon receipt of the ***inquire_DSG_mode()*** APDU, the Card SHALL send the ***set_DSG_mode()*** APDU as defined in Table 9.20-4 to inform the Host of the preferred operational mode for the network (either SCTE55_mode, advanced_DSG_mode or advanced_DSG_one-way_mode). The Card MAY send the ***set_DSG_mode()*** APDU as an unsolicited message at any time to the Host after the DSG resource session has been established. The method by which the Card determines the preferred operational mode is proprietary to the CA/Card system vendor.

In advanced_DSG_mode or advanced_DSG_one-way_mode the Host receives MPEG flows directly via DSG packets or indirectly thru the Extended Channel.

A Card should support a fall-back operational mode for cases where the Card is unable to obtain the preferred operational mode or the Host does not support the preferred operational mode. There are two potential default conditions that should be addressed:

- The Card has not acquired the preferred operational mode from the network due to possible network errors.
- The Card has acquired the preferred operational mode from the network but the Host does not support the preferred mode.

If the Card has not acquired the preferred operational mode from the network due to possible network errors, the Card SHALL instruct the Host that the preferred operational mode is SCTE55_mode. If the Host does not support the preferred DSG operational mode, the Card MAY select any alternative DSG operational mode supported by the Host.

If the operational mode of the Host is any of the DSG modes, the Host SHALL deny any tune requests for any SCTE 55 operational mode tuners. In any DSG mode, the Host SHALL disable the reverse QPSK transmitter for the QPSK RDC. In any DSG one-way modes, the Host SHALL disable the reverse eCM transmitter for the DOCSIS return channel.

Table 9.20-4 - set_DSG_mode APDU Syntax

Syntax	No. of Bits	Mnemonic

set_DSG_mode() {
  set_DSG_mode_tag
  length_field()
  operational_mode
}
	
	24
	
	8
		
		uimsbf
		
		uimsbf

set_DSG_mode_tag 0x9F9101

operational_mode Defines the preferred operational mode of the network.

0x00 SCTE55_mode - In this mode field, the reverse QPSK transmitter is under control of the Card through the use of the ***OOB_TX_tune_req()*** APDU in the Host Control resource. The Host responds to ***OOB_TX_tune_req()*** APDU, operational_mode field set to 0x00, by tuning the reverse QPSK transmitter to the requested frequency and coding value (bit-rate and power level). The Card uses the QPSK-RDC for returning data to the cable headend.

0x01 Reserved

0x02 Reserved

0x03 advanced_DSG_mode - In this mode, the Host uses either the eCM as the transmitter for the reverse path or, if operating as an SEB Client, uses the home network interface for the reverse path. If the Card attempts to command the reverse QPSK transmitter with the ***OOB_TX_tune_req()*** message while the Host is operating in the DSG mode, the Host denies the tune request with a "Tuning Denied - RF Transmitter busy" status. Also, in this mode, the receiver for the QPSK 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 denies the tune request with a "Tuning Denied - Other reasons" status. Setting this mode is equivalent to the state Notification from the DSG Client Controller: enable upstream transmitter defined in the DSG specification.
Note: In advanced_DSG_mode, broadcast messages (e.g., SCTE 65 SI messages, SCTE 18 EAS messages, OC Signaling) MAY be received by the Host directly via DSG Broadcast Tunnels or MAY be transmitted to the Host over the Extended Channel, as indicated in the ***DSG_directory()*** APDU.

0x04 advanced_DSG_one-way_mode - In this mode, the reverse QPSK transmitter and eCM Transmitter are disabled for both the QPSK RDC and the DOCSIS return channel. Also, in this mode, the receiver for the QPSK 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 denies the tune request with a "Tuning Denied - Other reasons" status. If the Card attempts to command the reverse QPSK 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 interrupts all two-way IP connectivity until another mode is selected.
Note: In advanced_DSG_one-way_mode, broadcast messages (e.g., SCTE 65 SI messages, SCTE 18 EAS messages, CVTs and OCAP XAITs) may be received by the Host directly via DSG Broadcast Tunnels or may be transmitted to the Host over the Extended Channel, as indicated in the ***DSG_directory()*** APDU.

05-0xFF Reserved

9.20.4 send_DCD_info APDU

The Host SHALL send the ***send_DCD_info()*** APDU as defined in Table 9.20-5 to pass DCD message TLV information to the Card. In DSG Advanced mode, the Host will reassemble DCD fragments, if necessary, and use the ***send_DCD_info ()*** APDU to pass the TLV-encoded data to the Card. If the Host receives the DCD message

from the eCM in 2 or more DCD fragments, the Host SHALL combine all DCD fragments, remove the DOCSIS MAC management header and the three header bytes (Configuration Change Count, Number of Fragments and Fragment Sequence Number) from each of the fragments, and send just the TLVs to the Card in the ***send_DCD_info()*** APDU. The Host uses the ***send_DCD_Info()*** APDU when the initial DCD for the current downstream channel is reassembled to send the information to the Card and subsequently when the Configuration Change Count is modified, in the event of an eCM MAC layer re-initialization or after a change to the Primary Downstream Channel. The Host SHALL send the ***send_DCD_Info()*** APDU to the Card when the initial DCD for a new downstream channel is reassembled after the eCM acquires a new Primary Downstream Channel. Upon receipt of the ***send_DCD_info ()*** APDU, the DSG Client Controller SHALL parse the DCD information, and, if there is any change to previously delivered filters, the Card SHALL send a new ***DSG_directory() APDU***.

Table 9.20-5 - send_DCD_info APDU Syntax

Syntax	No. of bits	Mnemonic

send_DCD_info () {
  send_DCD_info _tag
  length_field()
  DCD_message
}
	24
	
	(*)
		uimsbf

send_DCD_info_tag 0x9F9105

DCD_message The TLVs comprising the DCD message as defined in [DSG] in the Summary of DCD TLV Parameters table.

9.20.5 DSG_directory APDU

The Card SHALL send the ***DSG_directory()*** APDU as defined in Table 9.20-6 to provide DSG filter parameters to the eCM when operating in any Advanced DSG mode and the Host has reported resource DSG (0x00040041). The ***DSG_directory()*** APDU is sent either in response to the ***send_DCD_info()*** APDU or a ***DSG_message()*** APDU or may be an unsolicited APDU from the Card. The Card SHALL include in the ***DSG_directory()*** APDU all of the client IDs and associated DSG filters that may be released to the Host as determined by the Card, in addition to DSG filters associated with data flows to the Card.

The list of DSG filters provided in the ***DSG_directory()*** APDU overrides all previously defined DSG filters passed by the Card.

- If a DSG filter designates any specific layer-3/layer-4 parameters, then the eCM in the Host SHALL use the dsg_mac_address field and the specific layer-3/layer-4 parameters designated in the DSG filter to identify matching packets.
- If a DSG filter designates the entire UDP port range, then the eCM in the Host SHALL ignore layer-4 characteristics when identifying matching packets.
- If a DSG filter does not designate specific layer-3/layer-4 parameters (i.e., the DSG filter sets all values of Source IP address, Destination IP Address to 0, and sets the UDP ports to the entire range), then the eCM in the Host SHALL use only the dsg_mac_address value to identify matching packets.

When UCID is used as a classifier in a DCD rule, it is passed as a parameter in the ***DSG_directory()*** APDU. The Host uses the UCID acquired from the eCM as a match on the UCID contained in a directory entry to determine which DSG Filters to forward to the eCM. When no UCID matches occur, it needs to use the entry containing the default UCID = 0x00 in the ***DSG_directory()*** APDU. If the Host has not acquired a UCID in 2-way mode or is running in one-way mode, it needs to use the entry containing the default UCID = 0x00 in the ***DSG_directory()*** APDU. As noted in [DSG], it is expected that every DCD message that includes DSG Rules using UCID as a classifier also includes an additional Rule, of lower priority, that does not use UCID as a classifier. If the DCD message includes a DSG rule that does not use UCID as a classifier, the Card SHALL include this DSG Rule as the directory entry containing the default UCID = 0x00. UCID operation is detailed in the following flow chart:

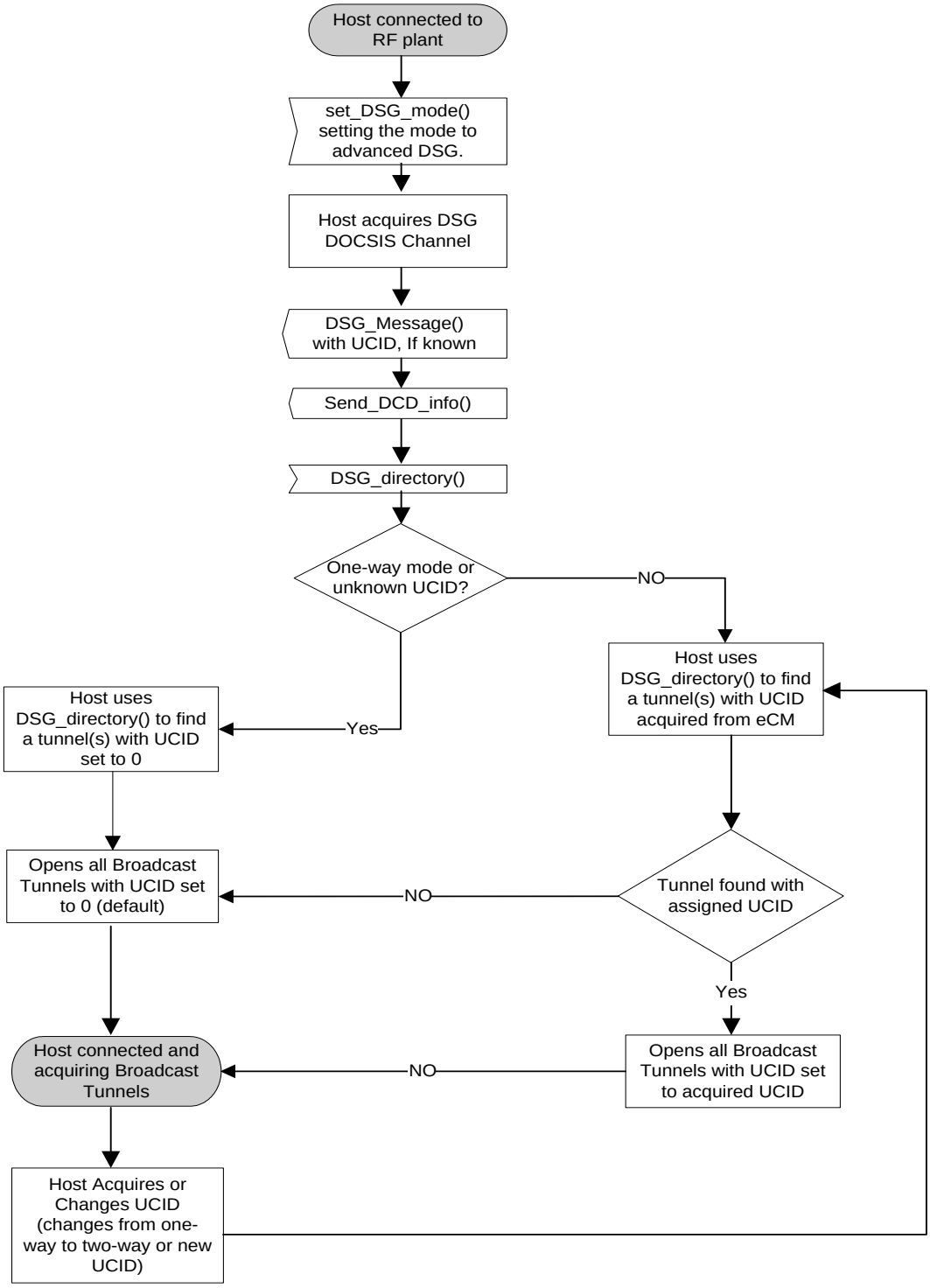


Figure 9.20-2 - UCID Flow Example from Host Perspective

If the vct_id_included field is set to 1, the Card SHALL provide a vct_id in the ***DSG_directory()*** APDU sent to the Host. This vct_id overrides any previously sent vct_id unless the vct_id is sent by the ***feature_parameters()*** APDU. When the Host is reinitialized it will revert to the default vct_id value of zero (0). The Card resends the ***DSG_directory()*** APDU to set the vct_id after the Host is reset and the vct_id value is known. As detailed in the following flow chart:

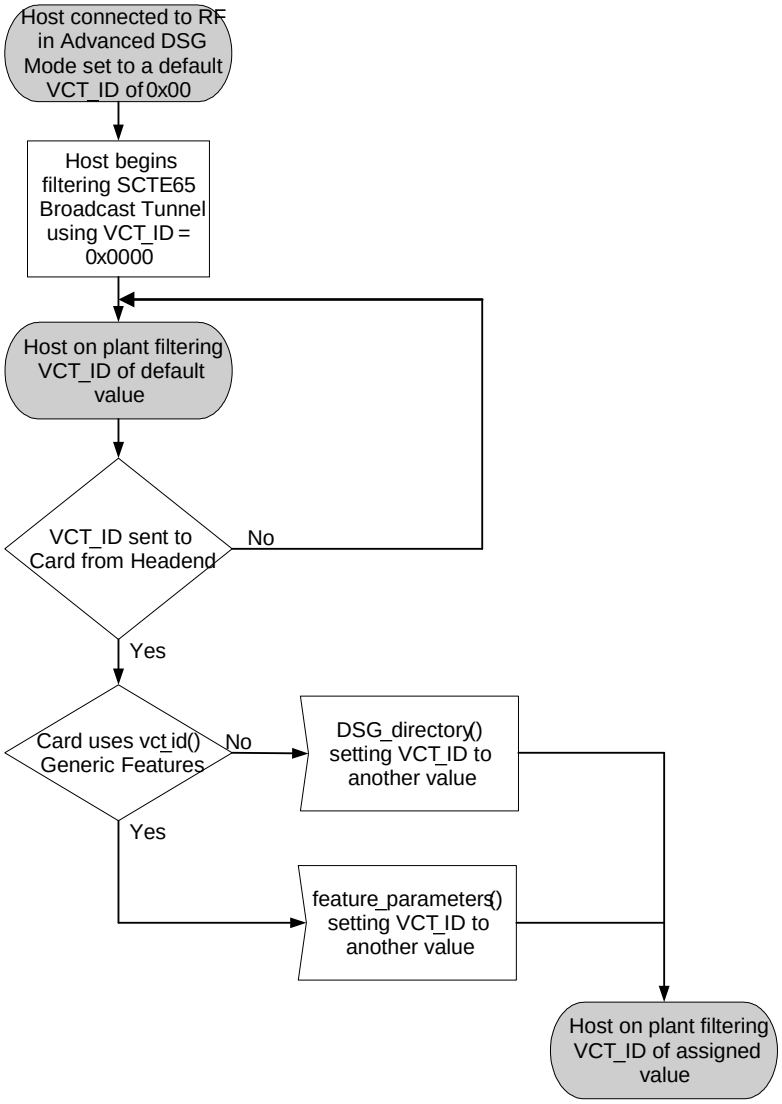


Figure 9.20-3 - VCT_ID Flow from Host Perspective

In any Advanced DSG mode, the Host eCM SHALL forward IP packets whose MAC destination address and layer-3/layer-4 parameters match any classifiers passed in the ***DSG_directory()*** APDU. The Host will determine which DSG filters to forward to the eCM based on dsg_client_id specified in the Host Entries section of the ***DSG_directory()*** APDU. The Host will forward to the eCM all DSG Filters specified in the Card Entries section of the ***DSG_directory()*** APDU.

- The dsg_client_id is used to designate the kind of DSG Client associated with the DSG filter in the number_of_host_entries loop if dir_entry_type = 0x01.
- A dir_entry_type equal to 0x01 in the number_of_host_entries loop indicates DSG filters associated with a DSG Client ID that is available to the Host directly.
- The Host SHALL terminate all packets which match the ADSG_filter() settings sent in the ***DSG_directory()*** APDU if the corresponding dir_entry_type is equal to 0x01.
- The Host may not forward a particular DSG Filter to the eCM if the device does not recognize or is not interested in the dsg_client_id associated with the ADSG filter().

- If dir_entry_type = 0x02, the data type associated with the dsg_client_id is a signal to the Host that this Broadcast data type will be available over an Extended Channel MPEG flow and is not delivered directly in a DSG Broadcast tunnel. Note that the DCD message delivered to the Card is not required to include a DSG Broadcast Client ID (TLV 50.4.1) corresponding to the data type indicated by dir_entry_type = 0x02 and the associated dsg_client_id. It is the sole responsibility of the Card to determine whether the Host receives Broadcast data types directly in DSG tunnels or over the Extended Channel.
- The Host SHALL forward to the Card all packets which match the ADSG_filter() settings defined in the number_of_card_entries loop.
- When operating in any Advanced DSG mode the Card MAY provide up to eight unique Ethernet MAC addresses along with a set of DSG classifiers for its use.
- The Host SHALL NOT re-establish DSG filters in the eCM when the list of DSG filters in the DSG_directory() APDU has not changed. [1]

The Host may receive a DSG Directory message that defines Host Entries that contain multiple instances of SCTE-65 Broadcast Tunnel ADSG Filters. The Host is not expected to consume all data from the multiple instances of SCTE-65 Broadcast Tunnels, as there may be multiple instances of similar tables (e.g., a Network Text Table may be present in each SCTE-65 Broadcast Tunnel, where each table is different). In such a case, the Host is required to parse the multiple tunnels in search of the tunnel that contains the Virtual Channel Table as defined by the applicable vct_id (i.e., Card provided vct_id or default as per Figure 9.20-3.

The Host SHALL open all SCTE-65 Broadcast Tunnels listed in the ***DSG_directory()*** APDU, if the number_of_host_entries loop defines multiple instances of dsg_client_id of 0x01 0x02 0x00 0x01.

- The Host SHALL parse the multiple SCTE-65 Broadcast Tunnels for the presence of the Virtual Channel Table with the applicable vct_id (either default or provided by the Card) and use this table for channel navigation functions.
- The Host SHALL close all SCTE-65 Broadcast Tunnels that do not contain the Virtual Channel Table with the applicable vct_id.

[1] The directory_version number includes changes to the VCT_ID, so it is not an exclusive indicator of whether the DSG filters have changed.

Table 9.20-6 - DSG_directory APDU Syntax

Syntax	No. of Bits	Mnemonic
DSG_directory() {		
DSG_directory _tag	24	uimsbf
length_field()		
reserved	7	bslbf
vct_id_included	1	bslbf
directory_version	8	uimsbf
number_of_host_entries	8	uimsbf
for (i=0; i< number_of_host_entries; i++) {		
dsg_client_id	8*N	uimsbf
dir_entry_type	8	uimsbf
if (dir_entry_type == 0x01){		
/** Direct Termination DSG Flow **/		
ADSG_Filter()		
UCID	8	uimsbf
}		
if (dir_entry_type == 0x02){		
/** Extended Channel MPEG Flow **/		
}		
number_of_card_entries	8	uimsbf
for (i=0; i< number_of_card_entries; i++) {		
ADSG_Filter()		
}		
number_of_RXFrequency	8	uimsbf
for (i=0; i<number_of_RXFrequency; i++) {		
RXFrequency	32	uimsbf
}		
initialization_timeout	16	uimsbf
operational_timeout	16	uimsbf
two_way_retry_timeout	16	uimsbf
one_way_retry_timeout	16	uimsbf
if (vct_id_included == 0x01) {		
vct_id	16	uimsbf
}		
}		

DSG_directory_tag 0x9F9104

vct_id_included Indicates if the vct_id is included in this message. The vct_id is defined in [SCTE65].

- 0b The vct_id is not included in this message (No change to last vct_id sent to Host, if any).
- 1b The vct_id is included in this message.

directory_version A module 256 counter that changes anytime any of the parameters in the directory are modified from a previous directory.

The Card SHALL change the directory_version in the ***DSG_directory()*** APDU anytime any of the parameters in the directory are modified from a previous directory.

If a directory is received from the Card which has the same directory_version number as the previous directory received from the Card in the ***DSG_directory()*** APDU, the Host MAY treat the new directory as identical to the previous directory and not process it.

The Card MAY send a directory in the ***DSG_directory()*** APDU with a different directory_version even if the contents are the same as the previous directory.

number_of_host_entries The number of directory entries for Host use provided in this message.

dsg_client_id The TLV-encoded DSG Client ID value associated with the directory entry. The TLV encoded values conform to Client ID values allowed by [DSG]. A DSG Client ID type is always in the context of type 50.4.x, where x varies depending on the type of Client ID (see [DSG]). The encoding of each instance of the dsg_client_id field in this message has an implicit prefix type of 50.4, which is not present in the message and explicitly begins with the appropriate value for x, followed by the appropriate length and value. Example Client ID encodings for the dsg_client_id field are:

Broadcast Client ID for [SCTE65] = 0x01 0x02 0x00 0x01

Broadcast Client ID for SCTE 18 = 0x01 0x02 0x00 0x02

Broadcast Client ID for XAIT = 0x01 0x02 0x00 0x05

Well-Known MAC Address Client ID
= 0x02 0x06 0xAA 0xBB 0xCC 0xDD 0xEE 0xFF

CAS Client ID 0x0A0B = 0x03 0x02 0x0A 0x0B

Application Client ID 16 = 0x04 0x02 0x00 0x10

dir_entry_type Indicates the acquisition method for data associated with the client.

0x01 DSG Filter
0x02 Extended Channel MPEG Flow. The data flow associated with the client ID is accessed via Extended Channel MPEG Flow. The use of this type of entry is only defined for the Broadcast Client IDs for SCTE-65, SCTE-18, CVT and OCAP XAIT.
0x03-0xFF Reserved

The Card SHALL NOT provide a directory with a Client ID value being associated with both a Host-terminated DSG Filter and an Extended Channel MPEG Flow.

UCID Upstream Channel ID - The UCID value contained in the DSG Rule, otherwise set to 0x00. **Note:** When a Host is running in one-way mode or 2-way mode, but has not acquired a UCID, the Host will use the default value of 0x00. When a DCD rule is defined using UCID, a default rule not containing UCID should also be defined as defined in [DSG].

number_of_card_entries The number of directory entries provided in this message describing DSG packets to be forwarded to the Card.

number_of_RXFrequency The number of TLV channel list entries in this message.

RXFrequency The RXFrequency as defined in [DSG].

initialization_timeout DSG Initialization Timeout (Tdsg1). The timeout period for the DSG packets during initialization as defined in [DSG]. In the ***DSG_directory ()*** APDU, a value of zero in the initialization_timeout field indicates that the default value as defined in [DSG] is used.

operational_timeout DSG Operational Timeout (Tdsg2). The timeout period for DSG packets during normal operation as defined in [DSG]. In the ***DSG_directory ()*** APDU, a value of zero in the operational_timeout field indicates that the default value as defined in [DSG] is 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]. The valid range of values is 0 to 65535. A value of zero (0) indicates that the Host should continuously retry two-way operation.

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]. The valid range of values is 0 to 65535. A value of zero (0) indicates the Host should immediately begin downstream scanning upon a Tdsg2 timeout.

vct_id The vct_id to be used by the host to filter on the correct virtual channel map.

Table 9.20-7 - ADSG_Filter Syntax

Syntax	No. of Bits	Mnemonic
ADSG_Filter () {		
tunnel_id	8	uimsbf
tunnel_priority	8	uimsbf
dsg_mac_address	48	uimsbf
source_IP_address	32	uimsbf
source_IP_mask	32	uimsbf
destination_IP_address	32	uimsbf
destination_port_start	16	uimsbf
destination_port_end	16	uimsbf
}		

tunnel_id An identifier for the tunnel. This field should match the DSG Rule ID received in the DCD message for tunnel identifier. The tunnel_id is used by the eCM to populate the dsgIfStdTunnelFilterTunnelId MIB object.

tunnel_priority Indicates the priority of the Tunnel.

dsg_mac_address The DSG MAC address associated with the DSG filter.

source_IP_address The IP source address of the DSG filter to be used in layer 3 filtering. A value of all zeros implies all values of SourceIP Address, i.e., this parameter was not specified in the DCD message.

source_IP_mask The source IP mask of the DSG filter 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. When source_IP_address is present in the DCD message and source_IP_mask is not present, a value of all ones is to be used for source_IP_mask.

destination_IP_address The IP destination address of the DSG filter to be used in layer 3 filtering. A value of all zeros implies all values of the Destination IP Address, i.e., this parameter was not specified in the DCD message.

destination_port_start The beginning of the range of UDP Destination Port numbers of the DSG filter.

destination_port_end The end of the range of UDP Destination Port numbers of the DSG filter.

9.20.6 DSG_message APDU

If the operational mode is any advanced DSG mode, the Host SHALL use the ***DSG_message()*** APDU as defined in Table 9.20-8 to indicate

- the eCM has established two-way communication and is passing the UCID of the upstream channel.
- the eCM cannot forward 2-way eSTB/Card traffic due to restrictions.
- the eCM has entered One-Way mode.
- the eCM has done a complete downstream scan without finding a DCD message.

- the eCM has received a DCC-REQ message and is preparing to execute a Dynamic Channel Change.
- an event has occurred that required an eCM MAC layer re-initialization.

Table 9.20-8 - DSG_message APDU Syntax

Syntax	No. of bits	Mnemonic
DSG_message() {		
dsg_message_tag	24	uimsbf
length_field()		
message_type	8	uimsbf
if (message_type == 0x01) {		
UCID	8	uimsbf
}		
if (message_type == 0x04) {		
init_type	8	uimsbf
}		
if (message_type == 0x07) {		
disabled_forwarding _type	8	uimsbf
}		
}		

dsg_message_tag 0x9F9103

message_type Indicates the purpose of the message:

0x00 Reserved

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.

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.

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()*** APDU with message_type 0x01 "2-way OK, UCID" to indicate the upstream has been established.

0x05 eCM Reset - An event has occurred that requires an eCM MAC layer re-initialization. The Card needs to re-establish DSG tunnel filtering by sending the ***DSG_directory()*** APDU after it receives message_type 0x01 2-way OK, UCID. The DSG tunnel MAC addresses and DSG classifiers are obtained by parsing the next received DCD message.

0x06 Reserved

0x07 eCM cannot forward 2-Way traffic -the eCM is in the Operational state, but cannot forward 2-Way traffic because of provisioning limitations.

0x08-0xFF Reserved.

UCID the channel ID of the DOCSIS channel that the Host is using for upstream communication.

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.

disabled_forwarding _type Specifies what type of eCM provisioning limitations impact eCM 2-Way forwarding. Values below are bit fields that can be ORed to indicate multiple conditions.

- 0x01 Network access disabled (NACO=0)
- 0x02 Max CPE limit exhausted
- 0x04 Forwarding interface administratively down

Informative Note: 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.

9.20.7 DSG_error APDU

The Card SHALL use the ***DSG_error()*** APDU as defined in Table 9.20-9 to inform the Host of a byte count error or an invalid DSG channel.

Table 9.20-9 - DSG_error APDU Syntax

Syntax	No. of Bits	Mnemonic

DSG_error() {
  DSG_error_tag
  length_field()
  error_status
}
	
	24
	
	8
		
		uimsbf
		
		uimsbf

DSG_error_tag 0x9F9102

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 then acquires a new DCD on a different downstream and passes 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.

0x02-0xFF Reserved

9.21 Headend Communication Resource

The Headend Communication Resource is intended for the transfer of specific control messages from the Headend to the Host through the Card. Support for the Headend Communication Resource is mandatory by the Card in both M-Mode and S-Mode. Support for the Headend Communication Resource is mandatory in the Host. The Card SHALL establish at least one session for the Headend Communication Resource using the identifier defined in Table 9.21-1.

9.21.1 Headend Communication Resource Identifier

The Host SHALL support the Headend Communication Resource, using the resource identifier as defined in Table 9.21-1.

The Card SHALL support the Headend Communication Resource using the resource identifier as defined in Table 9.21-1.

Table 9.21-1 - Headend Communication Resource (Type 1 Version 1)

Resource	Mode	Class	Type	Version	Identifier (hex)
Headend Communication	S-Mode/M-Mode	44	1	1	0x002C0041

9.21.2 Headend Communication APDUs

The Headend Communication Resource APDU messages are as follows:

Table 9.21-2 - Homing Objects

APDU_tag	Tag value	Resource	Direction Host ↔ Card
host_reset_vector	0x9F9E00	Headend Communication	←
host_reset_vector_ack	0x9F9E01	Headend Communication	→

9.21.3 host_reset_vector

The Card SHALL send the ***host_reset_vector()*** APDU as defined in Table 9.21-3 to the Host to deliver a recovery message. The Card generally obtains the ***host_reset_vector()*** in its entirety from the Cable Headend. When this is not possible, the Card may choose to construct the vector by itself, however, only by using the information received from the Headend through the various control messages.

Table 9.21-3 - host_reset_vector (Type 1, Version 1)

Syntax	No. of Bits	Mnemonic
host_reset_vector(){		
host_reset_vector_tag	24	uimsbf
length_field()		
transaction_id	8	uimsbf
reset_vector(){		
reserved	56	0x00…
delay	8	uimsbf
reset_field(){		
reserved	8	0x00
reset_embedded_cable_modem	1	
reset_security_element	1	
reset_host	1	
reset_external_devices	1	
reserved	3	0x0
reset_all	1	
}		
restart_field(){		
reserved	6	0x00
restart_ocap_stack	1	
restart_all	1	
}		
reload_application_field(){		
reserved	6	0x00
reload_all_ocap_apps	1	
reload_ocap_stack	1	
}		
reload_firmware_field(){		
reserved	7	0x00
reload_host_firmware	1	
}		
storage_clearing_field(){		
reserved	10	0x000
clear_persistant_gen_feature_params	1	
clear_org.dvb.persistant_fs	1	
clear_cached_unbound_ocap_apps	1	
clear_registered_libraries	1	
clear_persistant_host_memory	1	
clear_security_element_passed_values	1	
clear_non_asd_dvr_content	1	
clear_asd_dvr_content	1	
clear_network_dvr_content	1	
clear_media_volumes_int_hdd	1	
clear_media_volumes_ext_hdd	1	
clear_GPFS_internal_hdd	1	
clear_GPFS_external_hdd	1	
clear_all_storage	1	
}		
}		
}		

host_reset_vector_tag 0x9F9E00

transaction_id An 8-bit value, generated by the Card, to be returned in the corresponding ***host_reset_vector_ack()*** from the Host. The transaction_id allows the Card to match the Host's acknowledgement with a specific ***host_reset_vector()*** that it transferred. The Card should increment the value, modulo 255, with every ***host_reset_vector()*** it sends.

delay An 8-bit unsigned integer representing the number of seconds the Host waits before processing and acting on the received Reset Vector.

host_reset_vector() A 128-bit field used to transfer a reset vector from the Card to the Host. The definitions for the various fields are provided in the CableLabs Host Reset Vector Recommended Practice document.

The Host SHALL wait the amount of time indicated in the delay field in the ***host_reset_vector()*** APDU before processing and acting upon the message. The only field in the ***host_reset_vector()*** APDU that the Host is required to act upon SHALL be the reset_host bit in the reset_field() group.

If the Host receives another ***host_reset_vector()*** APDU before the delay expires, it SHALL disregard the current message and restart the delay according to the new message.

9.21.4 host_reset_vector_ack

The Host SHALL reply to the ***host_reset_vector()*** APDU using the ***host_reset_vector_ack()*** APDU as defined in Table 9.21-4 before performing any other operation.

Table 9.21-4 - host_reset_vector_ack (Type 1, Version 1)

Syntax	No. of Bits	Mnemonic
host_reset_vector_ack(){		
host_reset_vector_ack_tag	24	uimsbf
length_field()		
transaction_id	8	uimsbf
}		

host_reset_vector_ack_tag 0x9F9E01

transaction_id The transaction_id value received in the corresponding ***host_reset_vector()*** APDU from the Card.

9.22 Host Addressable Properties

The Host Addressable Properties resource is intended to be used to coordinate values known by the Card operating in M-Mode associated with a Host device with values associated with an addressable XAIT or AIT, in order to target an application to a specific Host or group of Hosts.

An XAIT or AIT may include one or more addressable attributes that are signaled as security system attributes in a host attribute comparison object contained in the addressing descriptor. Support for the Host Addressable Properties resource is optional for the Card operating in M-Mode. The Card MAY support Host Addressable Properties resource as defined in Table 9.22-1. The Host device will send a ***host_properties_req()*** APDU after it receives an XAIT or AIT addressable attribute that is marked as a security attribute. The Host SHALL support the Host Addressable Properties resource as defined in Table 9.22-1. The Host SHALL disregard XAIT or AIT addressable attributes that are marked as security system attributes if the Card has not opened a session for the Host Addressable Properties resource.

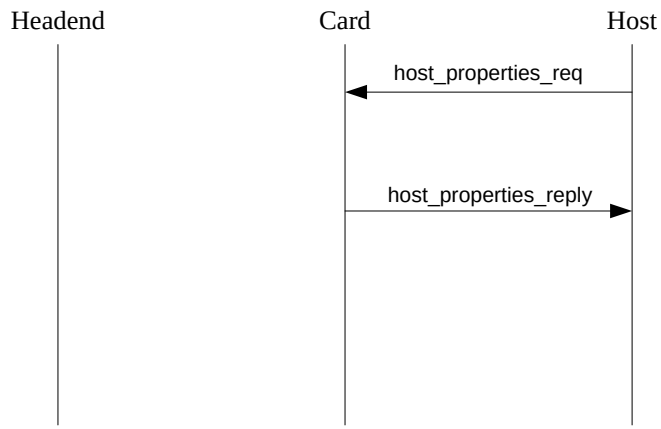


Figure 9.22-1 - Host Addressable Properties APDU Exchange

Table 9.22-1 - Host Addressable Properties Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Host Addressable Properties	M-Mode	256	1	1	0x01000041

9.22.1 Host Addressable Properties APDUs

The Host Addressable Properties Resource consists of the following two APDUs.

Table 9.22-2 - Host Addressable Properties APDUs

APDU Name	Tag Value	Resource	Direction Host ↔ Card
host_properties_req	0x9F9F01	Host Addressable Properties	→
host_properties_reply	0x9F9F02	Host Addressable Properties	←

9.22.1.1 host_properties_req() APDU

The Host SHALL send ***host_properties_req()*** APDU as defined in Table 9.22-3 if a session to the Host Addressable Properties resource session was opened and it determines an XAIT or AIT contains one or more addressable attributes that are marked as a security attribute. See [OCAP] chapter 11 Application Signaling for XAIT definition.

Table 9.22-3 - host_properties_req APDU syntax

Syntax	No. of Bits	Mnemonic
host_properties_req() {		
host_properties_req_tag	24	uimsbf
length_field()		
number_of_properties	8	uimsbf
for (i=0; i<number_of_properties; i++) {		
property_key_length	8	uimsbf
for (j=0; j<property_key_length; j++) {		
property_key_byte	8	uimsbf
}		
}		
}		

host_properties_req_tag 0x9F9F01

number_of_properties The number of properties queried by the Host device.

property_key_length Length of the property_key_char field.

property_key_byte The property_key field of the addressable property.

9.22.1.2 host_properties_reply() APDU

The Card SHALL send the ***host_properties_reply()*** APDU as defined in Table 9.22-4 in response to a ***host_properties_req()*** APDU. The Card SHALL compare the property key fields in the ***host_properties_req()*** APDU to determine if it supports any of the requested properties. For each property key entry in a ***host_properties_req()*** APDU, the Card SHALL include an identical property key entry in the ***host_properties_reply()*** APDU. If a match is found for a property key entry in the ***host_properties_req()*** APDU, the Card SHALL include an entry in the ***host_properties_reply()*** APDU with the property value corresponding to the property key. If a match is not found for a property key entry in the ***host_properties_req()*** APDU, the Card SHALL include an entry in the corresponding ***host_properties_reply()*** APDU with the property_value_length set to 0.

Table 9.22-4 - host_properties_reply APDU syntax

Syntax	No. of Bits	Mnemonic
host_properties_reply() {		
host_properties_reply_tag	24	uimsbf
length_field()		
number_of_properties	8	uimsbf
for (i=0; i<number_of_properties; i++) {		
property_key_length	8	uimsbf
for (j=0; j<property_key_length; j++) {		
property_key_byte	8	uimsbf
}		
property_value_length	8	uimbsf
for (j=0; j<property_value_length; j++) {		
property_value_byte	8	uimbsf
}		
}		
}		

host_properties_reply_tag 0x9F9F02

number_of_properties	The number of properties queried by the Host device.
property_key_length	Length of the property_key_byte field.
property_key_byte	The property_key field of the addressable property in UTF-8 format.
property_value_length	Length of the property_value_byte field.
property_value_byte	The property_value field of the addressable property in UTF-8 format.

9.23 Card MIB Access

The Card MIB Access resource allows the Host to access private MIB objects on the Card. Support for this Resource is mandatory for a Host device and optional for Cards.

The Host SHALL provide the Card MIB Access Resource using identifier(s) as defined in Table 9.23-1.

The Host SHALL keep the Card MIB Access Resource session open at all times during normal operation.

The Card MAY open a single session of the Card MIB Access Resource using identifier(s) defined in Table 9.23-1 after the CA Support Resource session initialization has completed.

If the Card opens the Card MIB Access Resource, it SHALL keep the session open at all times during normal operation.

When the SNMP agent on the Host receives an SNMP message from the DSG or FDC interface containing one or more OIDs that reside under the subtree defined by the response in ***get_rootOid_req()***, it checks the value of the Host MIB Card MIB Access Control Object (ocStbHostCardSnmpAccessControl). If the Control Object is set to TRUE, the SNMP agent generates an SNMPv2 message for each OID under the CableCARD subtree as shown in Table 9.23-3 and sends the request(s) to the Card via the ***snmp_req()*** APDU. The SNMP messages in the ***snmp_req()*** APDU must be in SNMPv2c format as specified in [RFC1901] and [RFC1902],

If the Card MIB Access Control Object is set to FALSE, the Host SNMP Agent SHALL ignore the Card OID subtree defined in the ***get_rootOid_req()*** APDU and reject any request for OIDs under this subtree with the SNMP exception response equivalent to the SNMPv2 "noSuchObject".

The Host will only send ***snmp_req()*** APDUs to the Card containing SetRequest, GetRequest, and GetNextRequest PDUs.

The Host may receive SNMP messages containing mixed OID requests, which are defined as messages specifying some OIDs destined for the Host and other OIDs destined for the Card.

The Host will parse mixed OID requests and generate individual SetRequest, GetRequest or GetNextRequest ***snmp_req()*** APDUs containing only those OIDs in the Card subtree, when the Card MIB Access Control Object is set to TRUE.

The Host will convert a GetBulkRequests into a series of one or more GetNextRequests ***snmp_req()*** APDUs containing only those OIDs in the Card subtree when the Card MIB Access Control Object is set to TRUE.

The Host SHALL parse the ***snmp_reply()*** APDU for the OID values and error indications to use in the SNMP Agent's response to the initial request.

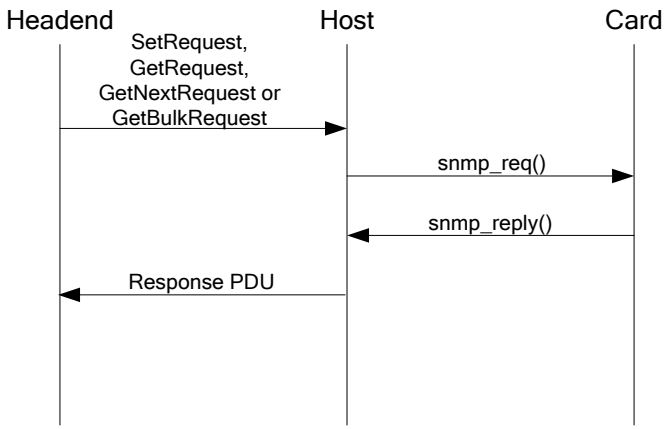


Figure 9.23-1 - Card MIB Access APDU Exchange

Table 9.23-1 - Card MIB Access Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Card MIB Access	M-Mode	90	1	1	0x005A0041

9.23.1 Card MIB Access APDUs

The Card MIB Access Resource consists of the following APDUs.

Table 9.23-2 - Card MIB Access APDUs

APDU Name	Tag Value	Resource	Direction Host ↔ Card
snmp_req()	0x9FA000	Card MIB Access	→
snmp_reply()	0x9FA001	Card MIB Access	←
get_rootOid_req()	0x9FA002	Card MIB Access	→
get_rootOid_reply()	0x9FA003	Card MIB Access	←

9.23.1.1 snmp_req() APDU

The Host SHALL send the ***snmp_req()*** APDU as defined in Table 9.23-3 with an OID that falls within the Card's subtree, as determined from the ***get_rootOid_req()*** APDU, to obtain MIB information from the Card. The Host SHALL wait for ***snmp_reply()*** APDU from the Card before sending another ***snmp_req()*** APDU.

Table 9.23-3 - snmp_req APDU syntax

Syntax	No. of Bits	Mnemonic

snmp_req() {
  snmp_req_tag
  length_field()
  for (i=0; i< N; i++) {
  snmp_message
  }
}
	24
	
	8
		uimsbf
		
		uimsbf

snmp_req_tag 0x9FA000

snmp_message An SNMP message with an OID that falls within the subtree belonging to the Card. SetRequest, GetRequest, and GetNextRequest are the only allowable messages.

9.23.1.2 snmp_reply() APDU

The Card SHALL send the ***snmp_reply()*** APDU as defined in Table 9.23-4 in response to a ***snmp_req()*** APDU.

Table 9.23-4 - snmp_reply APDU syntax

Syntax	No. of Bits	Mnemonic

snmp_reply() {
  snmp_reply_tag
  length_field()
  for (i=0; i< N; i++) {
  snmp_response
  }
}
	24
	
	8
		uimsbf
		
		uimsbf

snmp_reply_tag 0x9FA001

snmp_response The Response PDU to the snmp_message contained in the ***snmp_req()*** APDU.

9.23.1.3 get_rootOid_req()

If the Card has opened a session to the Card MIB Access resource, the Host sends a ***get_rootOid_req()*** APDU as defined in Table 9.23-5 to the Card.

Table 9.23-5 - get_rootOid_req() APDU Syntax

Syntax	No. of Bits	Mnemonic

get_rootOid_req() {
  get_rootOid_req_tag
  length_field() /* always = 0x00 */
}
	24	uimsbf

get_rootOid_req_tag 0x9FA002

9.23.1.4 get_rootOid_reply()

The Card SHALL send the ***get_rootOid_reply()*** APDU, as defined in Table 9.23-6, in response to the ***get_rootOid_req ()*** APDU from the Host.

The Card SHALL provide the root OID that provides a top level OID for all supported MIB objects in the ***get_rootOid_reply()*** APDU.

It is expected that the root OID will be in the vendor's SNMP private enterprise subtree.

Table 9.23-6 - get_rootOid_reply() APDU Syntax

Syntax	No. of Bits	Mnemonic

`get_rootOid_reply() {`
`  get_rootOid_reply_tag`
`  length_field()`
`  for (i=0; i< N; i++) {`
`  oid_byte`
`  }`
`}`
	`24`
	
	`8`
		`uimsbf`
		
		`uimsbf`

get_rootOid_reply_tag 0x9FA003

oid_byte Each oid_byte is an octet of ASCII characters that define the root OID of the Card.

10 EXTENDED CHANNEL OPERATION

The extended channel provides a data path between the Card and the Host used for network data. There are two possible states: SCTE 55 Mode (Card contains cable modem); and Advanced Mode DSG (Host device contains an eCM and processes network data directly or uses the extended channel for network data).

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.

Note (Informative): 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 or home network interface, if operating as an SEB Client, 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 via an IP_flow, 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 (or home network interface, if the Host is operating as an SEB Client). 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 or home network interface, if the Host is operating as an SEB Client, on behalf of the Card. The Host does not forward to the Card any datagrams received over other interfaces. When the Host is operating as an SEB Client, it only forwards to the Card datagrams received over that particular home network interface; all other packets from any other interface are to be discarded (e.g., if the Host is using the Ethernet port for SEB, then all packets addressed to the Card received from the USB port would be discarded).

The Host forwards all IP datagrams received from the Card to the eCM interface or home network interface, if operating as an SEB Client. 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. when it is not operating as an SEB Client. When the Host is operating as an SEB Client, it only forwards IP datagrams received from the Card to the interface being used for SEB (e.g., if the Host is using the Ethernet port for SEB, then all packets received from the Card are only forwarded to the Ethernet port), 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. Another example case where the flow may become unavailable is due to the SEB Server dropping from the home network, which is only applicable when the Host is operating as an SEB Client.

10.2 Socket Flows

When operating in DSG mode, an application on the Card has the ability to ask the Host to open a socket connection to communicate with a remote host. The socket connection can be either TCP or UDP. The Card has the option of requesting a specific local port on the Host or allowing the Host to choose an appropriate port for the local socket. The Card can open a socket connection instead of an IP flow; therefore, the Card does not need to obtain an IP address.

The Card resides at the Application Layer and the Host will utilize its eCM or home network interface, when operating as an SEB Client, to provide the Data Link Layer to the underlying DOCSIS network. The Host will use its IP stack to provide network layer services for the application on the Card. When the Card wishes to utilize the DOCSIS network to transfer IP datagrams upstream via a socket connection, it SHALL first submit a "new flow request" to the Host with a service type of 0x04 - socket.

The Host utilizes the socket opened in the socket flow request to transfer data between the remote destination and the Card. The Host extracts data from the IP datagrams bound for the Card's socket and forwards that data across the Extended Channel to the Card.

The Host forwards all data received from the Card over a socket flow to the eCM interface or home network interface, when operating as an SEB Client. 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., when it is not operating as an SEB Client. When the Host is operating as an SEB Client, it does not forward any IP datagrams received from the Card to any interface other than the home network interface (e.g., if the Host is using the Ethernet port for SEB, then all IP datagrams received from the Card are only forwarded to the Ethernet port).

If an established socket flow becomes unavailable for any reason, the Host SHALL report this to the Card using the ***lost_flow_ind()*** APDU. One example is when the remote Host in a TCP connection closes its socket.

10.3 Flow Examples—QPSK Modem Case

Figure 10.3-1 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).

As shown in Figure 10.3-1, 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. Both the Card and the Host are required to support at least one IP_U flow, but it is up to the applications in the Card and the Host as to whether an IP_U flow is opened.

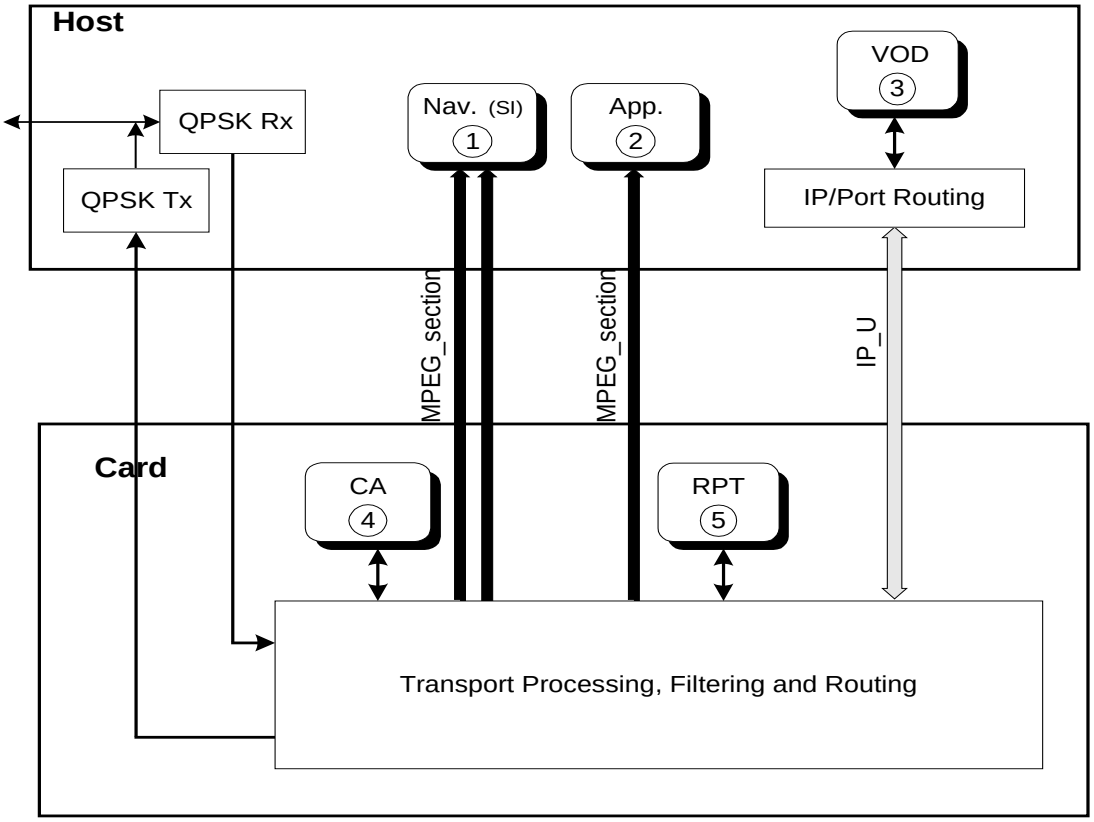


Figure 10.3-1 - Flow Examples - QPSK Modem Case

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.

10.4 Flow Examples—Embedded Cable Modem Case DSG Mode

In the next example case, the Host incorporates an eCM that supports Advanced DSG mode, but the Host does not directly terminate DSG Broadcast Tunnels. Figure 10.4-1 diagrams a CHI in which five flows have been set up. When a Host includes an eCM, it is required to support at least eight flows of service type Socket and one flow of service type DSG. In this example, the Card supports three MPEG section flows if the Host requests them.

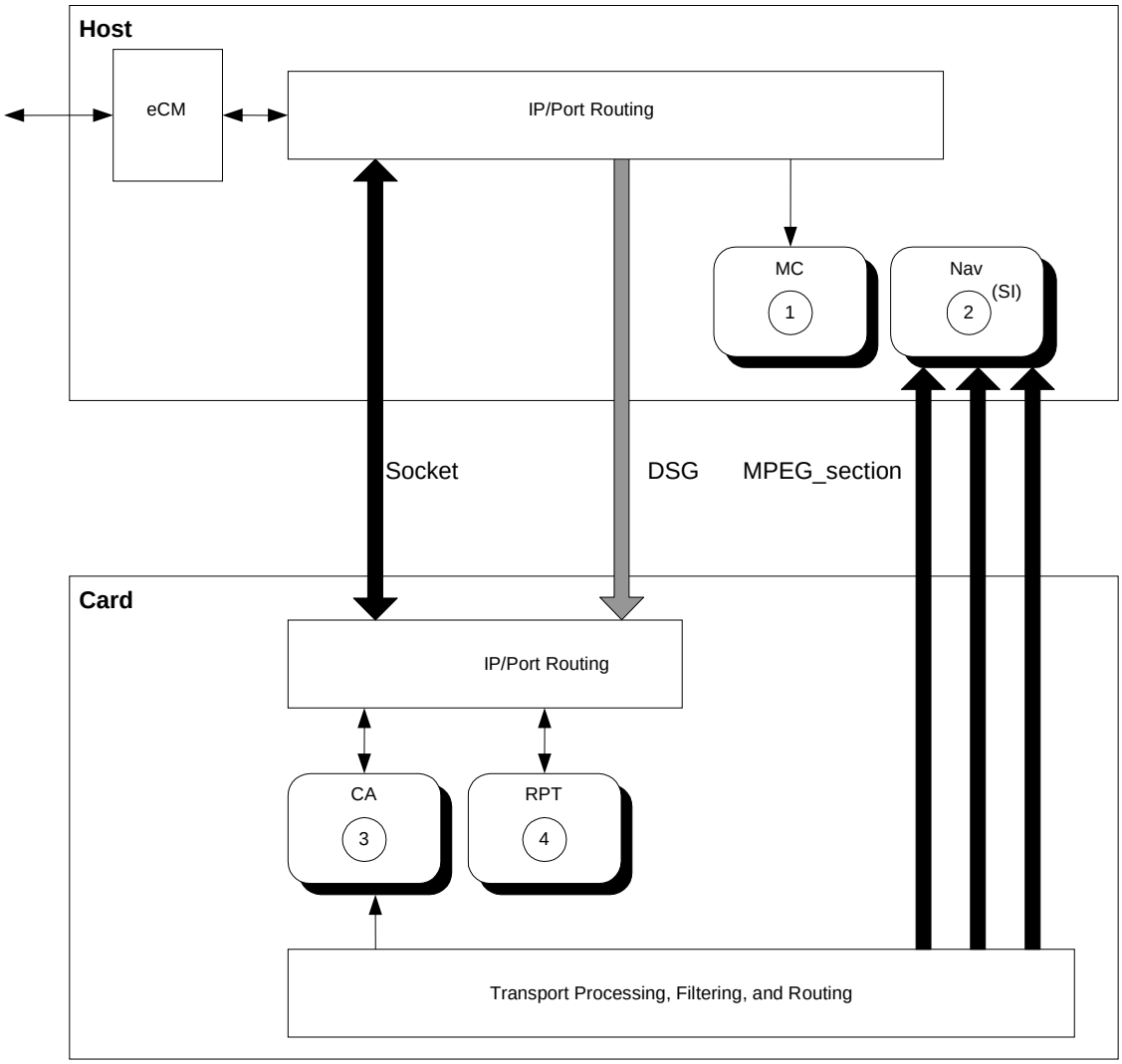


Figure 10.4-1 - Flow Examples - eCM Case Advanced Indirect Mode

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 delivers TCP/UDP packet payload data to the Card applications based on the port number specified in the Socket flow.

The Card includes a pay-per-view reportback function (#4) that uses standard TCP/UDP sockets for data transport.

In the following example, the Host incorporates an eCM that supports Advanced DSG mode, and the Host directly terminates DSG Broadcast Tunnels. Figure 10.4-2 diagrams a CHI in which 3 flows have been set up. In Advanced DSG mode, the Host may receive certain DSG Tunnels directly, without going through the Card.

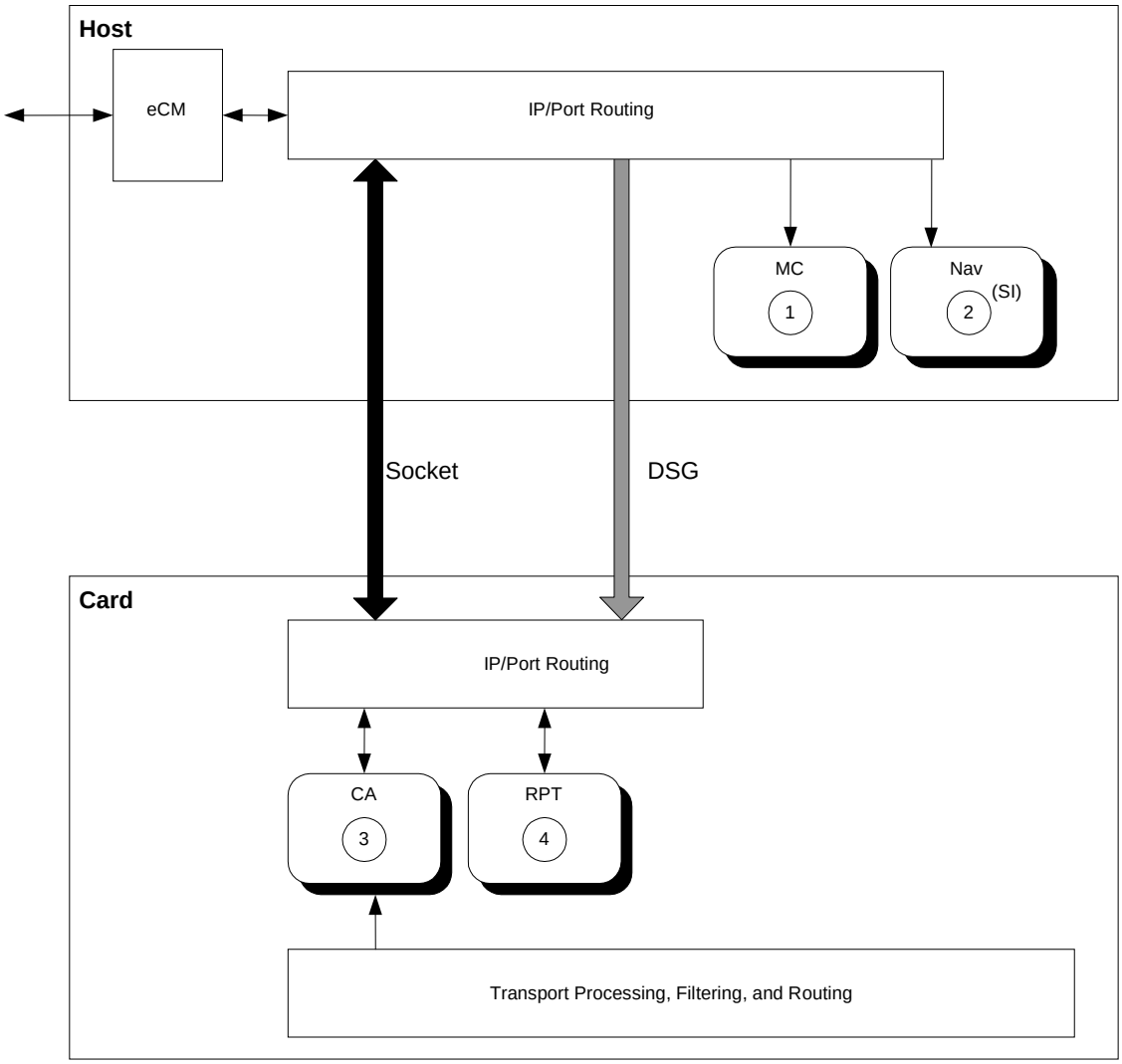


Figure 10.4-2 - Flow Examples - eCM Case Advanced Direct Mode

In this example, the Host has some application that uses multicast addressed packets (#1) and a Navigation application (#2) that receives Service Information directly data from the eCM.

The Host sets DSG filter(s) to the values associated with SCTE 65 Broadcast ID (see [DSG]) and the Navigation application would receive the tuning tables, Source Name Subtable, Virtual Channel Tables, and System Time. Additional Broadcast ID values are for SCTE 18, OC Signaling. The Card will also inform the Host of any other DSG Tunnels that it needs to receive.

If the Card includes applications that require TCP/UDP socket payload data, after opening a Socket flow, the Host will supply the payloads to the Card based on the port number.

In this example, the Card includes a pay-per-view reportback function (#4) that uses TCP/UDP sockets on the Host.

10.5 Flow Examples—SEB Client Case DSG Mode

In the next example case, the Host incorporates an eCM that supports Advanced DSG mode as well as SEB, as defined in [DSG], but the Host does not directly terminate DSG Broadcast Tunnels, and sends all DSG traffic across the CHI. Figure 10.5-1 diagrams a CHI in which five flows have been set up. The Host includes an eCM, which in this case continues to receive DSG Tunnels, but all IP data is acquired by the home network interface. The Host also

includes a home network interface (e.g., Ethernet port, MOCA, etc.) that is utilized to support SEB functionality, in this case to support SEB Client. The Host is required to support at least one flow of service type IP Unicast (IP_U) and one flow of service type DSG. In this example, the Card supports three MPEG section flows if the Host requests them.

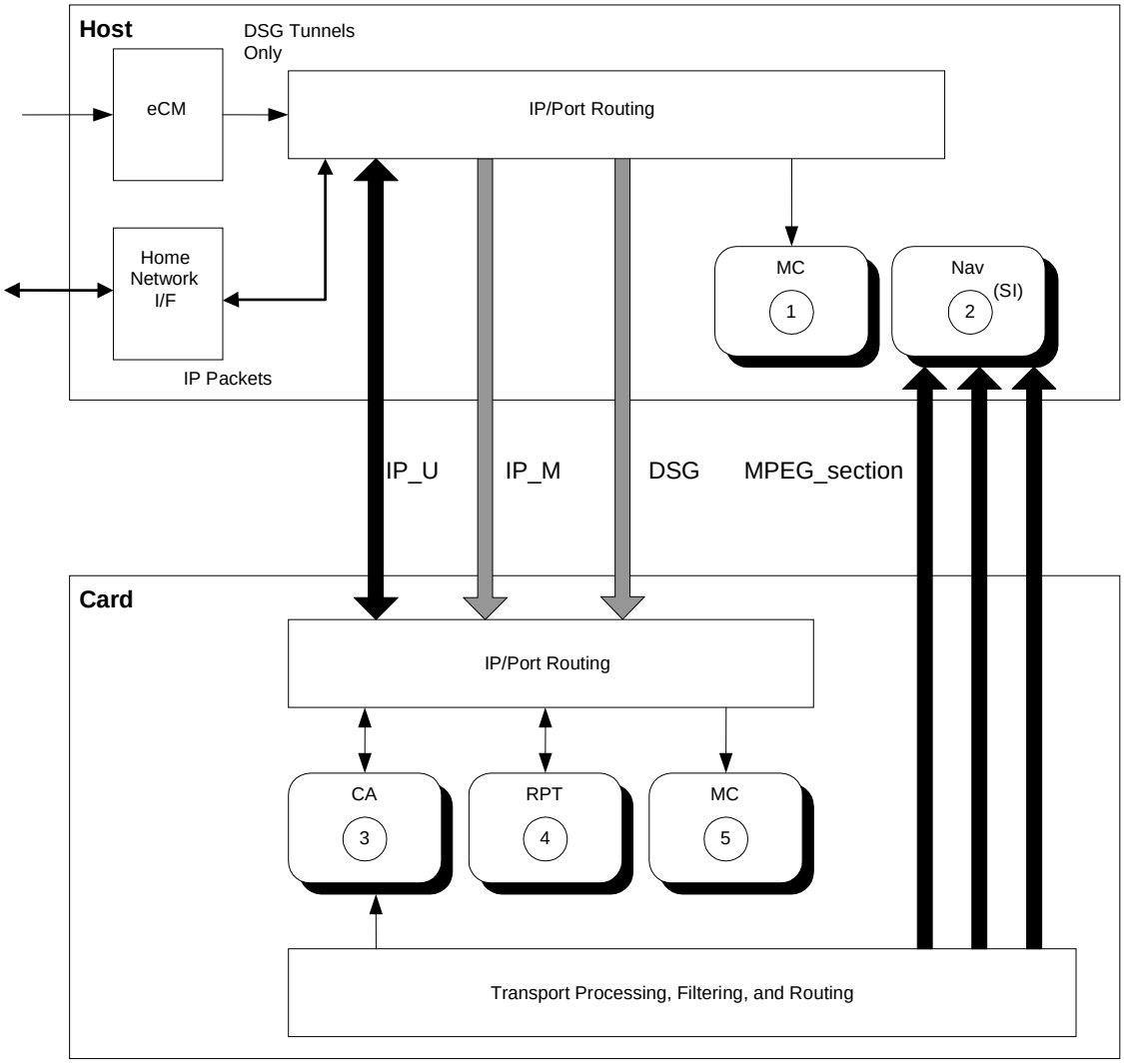


Figure 10.5-1 - Flow Examples - SEB Case Advanced Indirect Mode

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, which is serviced via proxy through the Host's home network interface.

In the following example, the Host incorporates an eCM that supports Advanced DSG mode as well as SEB, as defined in [DSG], and the Host directly terminates DSG Broadcast Tunnels. Figure 10.5-2 diagrams a CHI in which three flows have been set up. In Advanced DSG mode, the Host may receive certain DSG Tunnels directly, without going through the Card. The Host includes an eCM, which in this case continues to receive DSG Tunnels, but all IP data is acquired by the home network interface. The Host also includes a home network interface (e.g., Ethernet port, MOCA, etc.) that is utilized to support SEB functionality, in this case to support SEB Client.

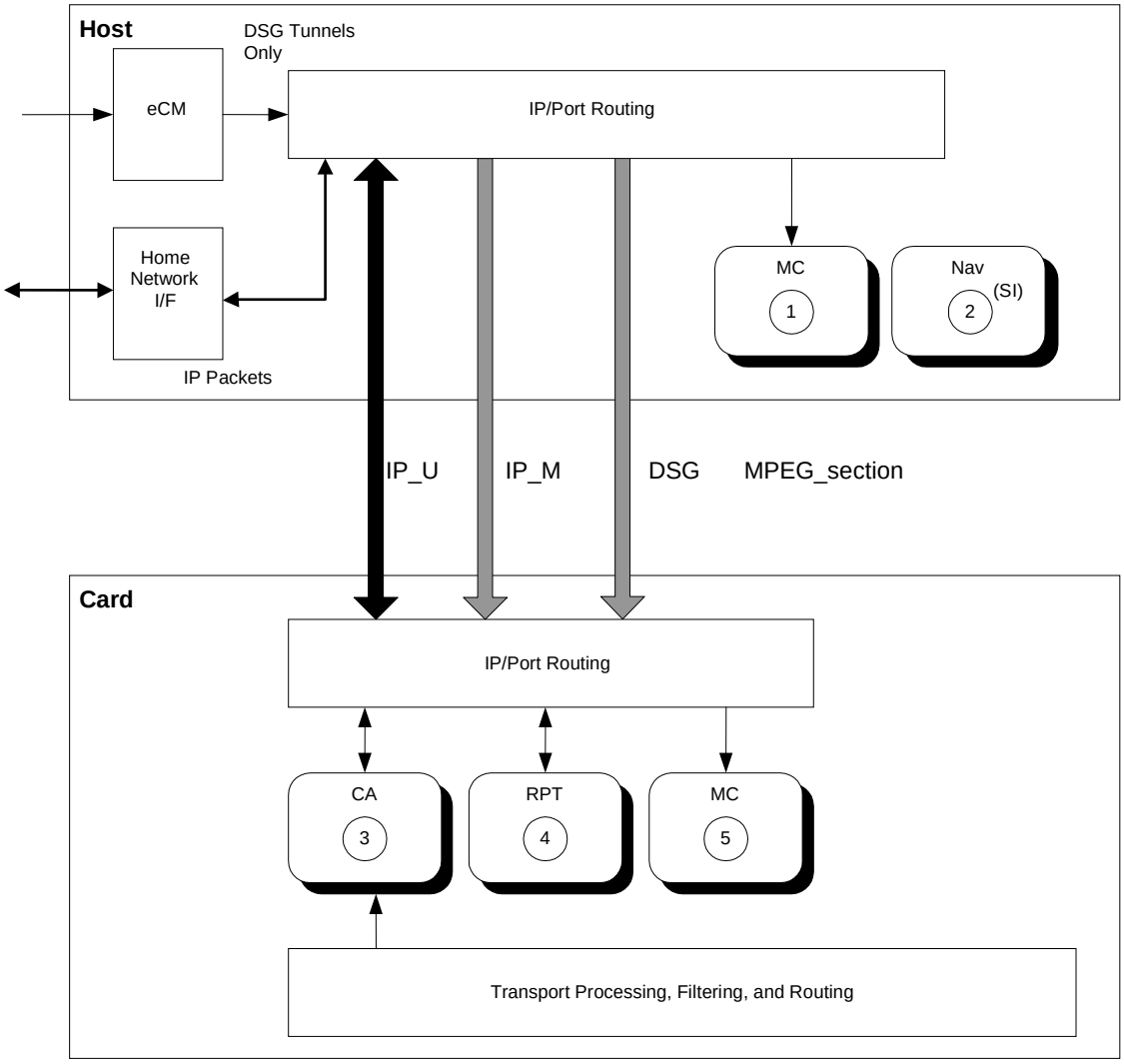


Figure 10.5-2 - Flow Examples - SEB Case Advanced Direct Mode

In this example, the Host has some application that uses multicast addressed packets (#1) and a Navigation application (#2) that receives Service Information directly data from the eCM.

The Host sets DSG filter(s) to the values associated with SCTE 65 Broadcast ID (see [DSG]) and the Navigation application would receive the tuning tables, Source Name Subtable, Virtual Channel Tables, and System Time. Additional Broadcast ID values are for SCTE 18, OC Signaling. The Card will also inform the Host of any other DSG Tunnels that it needs to receive.

If the Card includes applications that require IP, after opening an IP flow, the Host will supply the IP packets to the Card 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.

In this example, the Card includes a pay-per-view reportback function (#4) that uses standard IP packets. Finally, the Card includes some application (#5) that has registered with the Host to receive multicast-addressed IP packets through the home network interface via SEB from the SEB Server.

10.6 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 10.6-1 - Flow Requirements

Operational Mode	Service Type	Requestor	Minimum Number of Possible Concurrent Flows Supported	Data Direction
SCTE55	MPEG	Host	6	Card → Host
	IP_U	Host	1	Host ↔ Card
	IP_M	Host	1 (Optional)	Card → Host
	DSG	N/A	0	N/A (Error)
	Socket	N/A	0	N/A (Error)
Advanced DSG (Indirect)	MPEG	Host	6	Card → Host
	IP_U	Card	0	N/A (error)
	IP_M	Card	0	N/A (error)
	DSG	Card	1	Host → Card
	Socket	Card	8	Host ↔ Card
Advanced DSG (Direct)	MPEG	N/A	0 (if all Client IDs direct)	N/A (Error if all Client IDs direct)
	IP_U	Card	0	N/A (error)
	IP_M	Card	0	N/A (error)
	DSG	Card	1	Host → Card
	Socket	Card	8	Host ↔ Card

10.7 System/Service Information Requirements

When the operational_mode in the ***set_DSG_mode()*** APDU is ≤0x02, the Card supplies System and Service Information across the HOST- Card interface, using Extended Channel flow of ***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 conforms to one or more of the profiles specified in [SCTE65].

When the operational_mode in the ***set_DSG_mode()*** APDU is 0x03 or 0x04, the Card supplies the DSG MAC address and filter parameters to allow the Host to receive the System and Service Information directly via DSG or across the Host/Card interface, using an Extended Channel flow of service_type = MPEG_section. The set of

MPEG-2 tables provided to support the navigation function of the Host device conform to one or more of the profiles specified in [SCTE65].

SI data is transmitted either from the Card to the Host using the protocols defined in [SCTE65], or is directly received by the Host using the protocols defined in [SCTE65]and [DSG]. The Card MAY be the source of all SI data to the Host. The Card MAY reformat the SI data received over the network to meet the requirements of [SCTE65] if such data will be sent to the Host via the Extended Channel, as there is no requirement for the cable system to transmit the SCTE 55 SI data in that format.

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].

10.8 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.

10.8.1 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 10.8-1 - 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.

10.8.2 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 10.8-2 - 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 are 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) is 4,096 for any Service Type.

10.9 Modem Models

There are 4 different network connection models that a Host may have:

- Unidirectional (no modem)
- Bidirectional, modem function in the Card
- Bidirectional, modem function in the Host
- Bidirectional, SEB function in the Host

10.9.1 Unidirectional Host Model

For the unidirectional Host model, there are no upstream transmitters; thus, 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.

10.9.2 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 5.11.1 of this specification.

For this model, the Card will be the link device for all flows.

10.9.3 Bidirectional With Modem in Host

When the Host implements the DSG functionality, all of the DOCSIS cable modem functionality is 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.

10.9.4 Bidirectional With SEB in Host

When the Host implements the SEB functionality, all of the DSG cable modem functionality and non-DSG related IP data functionality are implemented in the Host via SEB. The OOB data flows are transmitted, utilizing DSG tunneling [DSG] SEB that continues to be terminated by the eCM in the Host. 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.

10.10 Section removed (duplication with section 10.7)

10.11 EAS Requirements

The Card will receive Emergency Alert messaging on either the FAT channels, the QPSK Forward Data channel (QPSK FDC), or over a DSG tunnel. The EAS message syntax is compatible with MPEG-2 transport and is defined in [J042].

For FAT channel transmission, the EAS message appears in transport packets with the same PID as those used for Service/System Information (SI), and the Card SHALL NOT interfere when delivering the EAS message to the Host. The table ID for the EAS message is 0xD8 as defined in [J042].

For SCTE 55 mode, the Card SHALL process EAS messages and transmit the EAS messages over the Extended Channel according to [J042].

In Advanced DSG Mode, the Host SHALL process EAS messages directly with no assistance by the Card if this tunnel is defined by a DSG Filter in the host section (dir_entry_type = 0x01) of the ***DSG_directory()*** APDU.

In Advanced DSG Mode, the Host SHALL receive EAS messages over the Extended Channel when indicated by dir_entry_type = 0x02 in the host section of the ***DSG_directory()*** APDU.

EAS messages can be transmitted over the DSG tunnel defined by Broadcast ID 0x02 (see [DSG]) using the protocol defined in [J042]. When a Card is installed in the Host and the operational_mode is SCTE 55 Mode, the Host SHALL only respond to EAS messages delivered from the Card over the Extended Channel. The Card may reformat the EAS message to meet the requirements of [J042], as there is no requirement for the cable system to transmit the 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.

10.12 XAIT Requirements

When operating in SCTE55_Mode, the Card SHALL forward all received XAIT messaging across the extended channel as defined in the OpenCable Application Platform Specification [OCAP]. When operating in Advanced DSG mode, the Host SHALL receive XAIT messages over the DSG Broadcast tunnel defined for XAITs if this tunnel is defined by a DSG Filter in the host section (dir_entry_type = 0x01) of the ***DSG_directory()*** APDU. When operating in Advanced DSG mode, the Host SHALL receive XAITs over the Extended Channel when indicated by dir_entry_type = 0x02 in the host section of the ***DSG_directory()*** APDU.

10.13 OCAP OOB Object Carousel Requirements

When the Host is in Advanced DSG Mode, it SHALL NOT open any flows over the Extended Channel for the OCAP object carousel.

When the operational mode is SCTE 55 Mode, the Host MAY open a flow over the extended channel for the OCAP object carousel. If the Host does open a flow over the Extended Channel for an OCAP object carousel, it SHALL first open an MPEG flow to PID 0x0000 to retrieve the PAT and then open an MPEG flow to the PMT PID defined in the PAT. If the PMT defines the OCAP object carousel to be on the same PID as the PMT, the Host SHALL NOT open a new flow to that PID (as the flow is already open). If the PMT defines the OCAP object carousel to be on a different PID, then the Host SHALL open an MPEG flow to that PID to receive the object carousel. While receiving the object carousel, the Host SHALL keep the MPEG flows for the PAT and PMT open and accept any changes in those tables and act upon those changes.

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 " 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 ".

"
"
"

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		
	Line feed
	Space	&#32	
!	Exclamation mark	!	
"	Quotation mark	"	"
#	Number sign	#	
$	Dollar sign	$	
%	Percent sign	%	
&	Ampersand	&	&
'	Apostrophe	'	
(	Left parenthesis	(	
)	Right parenthesis	)	
*	Asterisk	*	
+	Plus sign	+	
,	Comma	,	
-	Hyphen	-	
.	Period	.	
/	Solidus (slash)	/	
0		0	

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		e	
f		f	
g		g	
h		h	
I		i	
j		j	
k		k	
l		l	
m		m	
n		n	
o		o	
p		p	
q		q	
r		r	
s		s	
t		t	
u		u	
v		v	
w		w	
x		x	
y		y	
z		z	
{	Left curly brace	{	
|	Vertical bar			
}	Right curly brace	}	
~	Tilde	~	
	Non-breaking space		
¡	Inverted exclamation	¡	¡
¢	Cent	¢	¢
£	Pound	£	£
¤	Currency	¤	¤
¥	Yen	¥	¥
¦	Broken vertical	¦	¦
§	Section sign	§	§
¨	Umlaut/diaeresis	¨	¨
©	Copyright	©	©
ª	Feminine	ª	ª
«	Left angle quote	«	«
¬	No sign	¬	¬
-	Hyphen		
®	Reg. trade mark	®	®
¯	Macron	¯	¯
°	Degrees	°	°
±	Plus/Minus	±	±
²	Superscript 2	²	²
³	Superscript 3	³	³
´	Acute accent	´	´
µ	Micron	µ	µ
¶	Paragraph sign	¶	¶
·	Middle dot	·	·
¸	Cedilla	¸	¸
¹	Superscript 1	¹	¹

Character	Name	Numeric Expression	Mnemonic Expression
º	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

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
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.	S-Mode	None	Host reports error to user.
2	Host reads incorrect CIS values	Card	Host reports error using screen in Figure B-1 - Error Display.	S-Mode	None	Host reports error to user.[1]
3	Host writes incorrect TPCE_INDX value to POD configuration register	Host	None	S-Mode	Card cannot perform any action.	Host detects as failure #4 and reports error to user.[1]
4	Host sets command 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.	S-Mode	None	Host reports error to user.
5	Host sets command channel RS bit and 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.	S-Mode	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	S-Mode	None	Host reports error to user.[1]
7	Invalid buffer negotiation - Host data channel (buffer size < 256 bytes or greater than Card data channel buffer size)	Host	None	S-Mode	Minimum - Card sets IIR flag and stops responding to polls. Preferred - Card works with Host buffer size	Host reports error to user.[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
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	S-Mode	None	Host reports error to user.[1]
9	Invalid buffer negotiation - Host extended channel (buffer size < 256 bytes or greater than Card data channel buffer size)	Host	None	S-Mode	Minimum - Card sets IIR flag and stops responding to polls. Preferred - Card works with Host buffer size	Host reports error to user.[1]
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.	S-Mode	None	Host reports error to user.
11	Host does not respond to Card request to open resource manager session within 5 seconds	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
12	Host response to open resource manager session response - resource manager non-existent	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
13	Host response to open resource manager session response - resource manager unavailable	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
14	Host response to open resource manager session response - incorrect version of resource manager	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
15	Host response to open resource manager session response - resource manager busy	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
16	Host response to open resource manager session response - invalid status byte	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]

	Error Condition	Failure	Host Action	Card Mode	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.	S-Mode M-Mode	None	Host reports error to user.
18	Host resource response - no application information resource	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
19	Host resource response - no Host control resource	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Card may not be able to do conditional access properly.
20	Host resource response - no system time resource	Host	None	S-Mode M-Mode	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.[1]
21	Host resource response - no MMI resource	Host	None	S-Mode M-Mode	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.[1]
22	Host resource response - no low speed communications	Host	None	S-Mode M-Mode	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, and the unit may function as a uni-directional device.[1]
23	Host resource response - no homing resource[1]	Host	None	S-Mode M-Mode	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).[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
24	Host resource response - no copy protection resource	Host	None	S-Mode M-Mode	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.[1]
25	Host resource response - unknown resource identifier	Host	None	S-Mode M-Mode	Minimum - Card continues operation.	Not a failure condition
26	Host fails to respond to open session request within 5 seconds	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
27	Host response to open application info resource session - application info non-existent	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
28	Host response to open application info resource session - application info unavailable	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
29	Host response to open application info resource session - incorrect version of application info	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
30	Host response to open application info resource session - application info busy	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
31	Host response to open application info resource session - invalid status byte	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
32	Card requests to open conditional access session to the Host times out after 5 seconds	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	Host reports error to user.[1]
33	Card response to conditional access resource session - conditional access non-existent	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
34	Card response to conditional access resource session - conditional access unavailable	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
35	Card response to conditional access resource session - incorrect version of conditional access	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
36	Card response to conditional access resource session - conditional access busy	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
37	Card response to conditional access resource session - invalid status byte	Host	None	S-Mode M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. 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.[1]
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.	S-Mode M-Mode	None	Host reports error to user.
39	Card requests to open copy protection resource session to the Host times out after 5 seconds	Host	None	S-Mode M-Mode	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.[1]
40	Host response to open copy protection resource session - copy protection non-existent	Host	None	S-Mode M-Mode	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.[1]
41	Host response to open copy protection resource session - copy protection unavailable	Host	None	S-Mode M-Mode	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.[1]
42	Host response to open copy protection resource session - copy protection busy	Host	None	S-Mode M-Mode	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.[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
43	Host response to open copy protection resource session - invalid status byte	Host	None	S-Mode M-Mode	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.[1]
44	Host does not support the Card's copy protection system	Host/ Card incom-patibility	None	S-Mode M-Mode	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.[1]
45	Host and Card do not mate	Host/ Card incom-patibility	None	S-Mode M-Mode	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.[1]
46	Host response to CP_sync - Host busy	Host	None	S-Mode M-Mode	Minimum - Card will cease descrambling of copy protected channels.	A copy protected channel will stop being descrambled.
47	Host response to CP_sync - no CP support	Host	None	S-Mode M-Mode	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	S-Mode M-Mode	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	S-Mode M-Mode	Minimum - Card will cease descrambling of copy protected channels and, S-Mode, set the IIR flag. M-Mode, sets the ER bit in the IQB.	A copy protected channel will stop being descrambled.
50	Invalid Host certificate	Host	None	S-Mode M-Mode	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.[1]

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
51	Write Error (WE) occurs after completion of any transfer from Host to Card	Card or Host	Host performs Card reset.	S-Mode	None	User may see frozen picture on scrambled channels.[1]
52	Read Error (RE) occurs after completion of any transfer from Card to Host	Card or Host	Host performs Card reset.	S-Mode	None	User may see frozen picture on scrambled channels.[1]
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.	S-Mode M-Mode	None	User MAY see frozen picture on scrambled channels.
54	Invalid session APDU from Host	Host	None	S-Mode M-Mode	No action	Not a failure condition
55	Invalid session APDU from Card	Card	Host ignores invalid sessions.	S-Mode M-Mode	None	Not a failure condition
56	Invalid SPDU tag from Host	Host	None	S-Mode M-Mode	No action	Not a failure condition
57	Invalid SPDU tag from Card	Card	Host ignores invalid SPDU tags.	S-Mode M-Mode	None	Not a failure condition
58	Invalid APDU tag from Host	Host	None	S-Mode M-Mode	No action	Not a failure condition
59	Invalid APDU tag from Card	Card	Host ignores invalid APDU tags.	S-Mode M-Mode	None	Not a failure condition
60	Transport ID from Host that has not been created and confirmed by Card	Host	None	S-Mode M-Mode	No action	Not a failure condition
61	Transport ID from Card that has not been created by Host	Card	Host ignores transport IDs that have not been created	S-Mode M-Mode	None	Not a failure condition
62	Session ID from Host that has not been created and confirmed by Card	Host	None	S-Mode M-Mode	No action	Not a failure condition
63	Session ID from the Card that has not been created by Host	Card	Host ignores session IDs that have not been created	S-Mode M-Mode	None	Not a failure condition
64	Incompatible CableCARD device Inserted	Host	Reports error using screen in Figure B-2 - Error Code 161-64 Display	M-Mode	None	Used when an S-CARD in inserted into an M-Host.

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
65	Card Resource Limit Reached	Card	Reports error using screen in Figure B-1 - Error Display	M-Mode	None	Used when the stream, program and/or PID limit has been reached by a user initiated action.
66	When the Card is in M-Mode and the Host sets the ER bit but the Card fails to set the CR bit in the IQB within 5 seconds of RESET going inactive	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.	M-Mode	None	Host reports Error to user
67	Host resource response - no Extended Channel resource	Host	None	S-Mode/ M-Mode	Minimum - Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB. Preferred - Card continues operation and will not open a session to the Extended Channel resource.	Minimum - Host reports error to user. Preferred - Applications on the Card and/ or Host may not operate correctly.
68	Host resource response - no System Control Resource	Host	None	S-Mode/ M-Mode	Minimum - Card continues operation and will not open a session to the System Control resource. Preferred - Same as minimum but also reports this in its MMI diagnostics application.	Minimum - Host reports error to user.[1] Preferred - Common Downloads to the Host may not function properly.
69	Host resource response - no CARD RES Resource	Host	None	M-Mode	Minimum - Card continues operation and will not open a session to the CARD RES resource. Preferred - Same as minimum but also reports this in its MMI diagnostics application	Minimum - Host reports error to user.[1] Preferred - Interface limits may be reached and the Host may not function properly, and/ or may also display error code 65.
70	Host resource response - no DSG Resource	Host	None	M-Mode	Minimum - Card continues operation and will not open a session to the DSG resource. Preferred - Same as minimum but also reports this in its MMI diagnostics application.	Host reports error to user.[1] Preferred - DSG operations/ messaging to the Card/Host may not function properly.

	Error Condition	Failure	Host Action	Card Mode	SCTE Card Action	Comments
71[2]	The M-CARD in M-Mode failed to open a Resource Manager session within 10 seconds after the RESET went inactive	Card	If the Host supports this optional feature, it performs a PCMCIA reset upon detection of the error condition. Retry one additional PCMCIA reset if still not successful. If the two PCMCIA resets do not clear the problem, power cycle the Card by powering down the V_{CC} pin. During power down, V_{IH} must be less than V_{CC} + 0.5V. After a 5-second wait time, re-apply power to the V_{CC} pin as defined in Section 7.4.1.2. Retry one additional power cycle if still not successful.	M-Mode Only	None	Optional: Host reports error to user when the two PCMCIA resets and the two power cycles do not clear the problem.
72[2]	Host fails to send profile_inq() APDU within 5 seconds of Resource Manager session being established	Host	None	S-Mode M-Mode	Card, S-Mode, sets IIR flag and stops responding to polls. M-Mode, sets the ER bit in the IQB.	None
73	Host resource response - no Headend Communication Resource	Host	None	S-Mode M-Mode	Minimum - Card continues operation.	Host reports error to user.

[1] - If the error is caused by an issue with the design of the Host or Card, this should be detected during certification.

[2] - Errors #71 and #72 are optional.

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.

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:

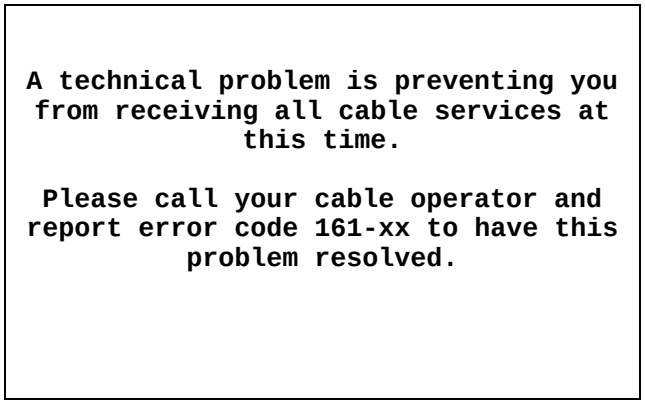


Figure B-1 - Error Display

The “xx” after the error code 161 SHALL be the item number of the above table which has failed.

For error code 161-64, which occurs when an S-Card or PCMCIA module is inserted into a Host supporting M-Mode only, the following message, or its equivalent, SHALL be displayed: "An incompatible Module has been inserted (error code 161-64). Please call your cable operator and request an M-CARD."

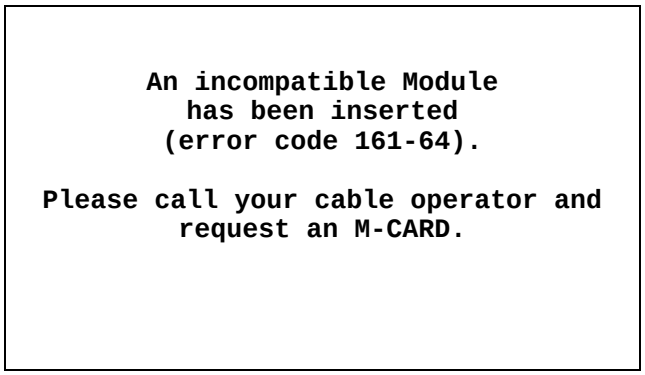


Figure B-2 - Error Code 161-64 Display

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.

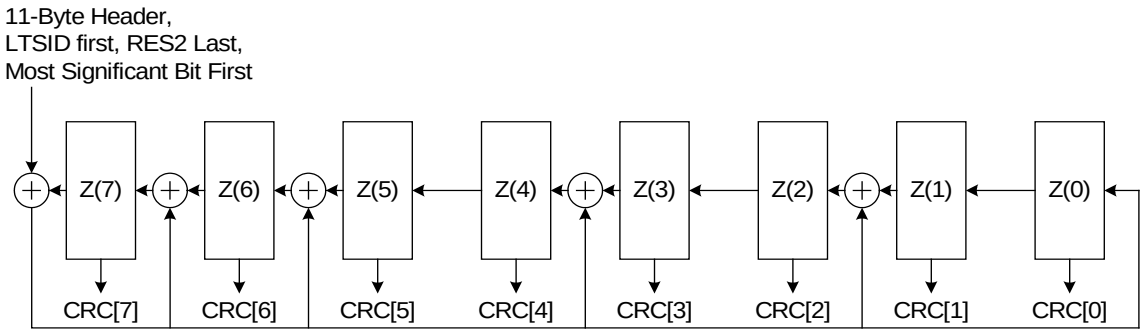


Figure C-1 - 8 bit CRC generator/checker model

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 12th byte of the MPEG Transport Stream Pre-Header (for the generator) or compared with the 12th 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$_{(hex)}$	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_0A

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	TPLLV1_MAJOR = 0x05							
3	06	TPLLV1_MINOR = 0x00							
4	08	TPPLV1_INFO = {Name of manufacturer (n bytes)							
4+n	08+(2*n)	TPLLV1_INFO (multiple bytes) ox00 (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$_{(hex)}$	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$_{(hex)}$	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$_{(hex)}$	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_INDX has both bits 6 (Default) and 7 (Intface) 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 Vcc is modified to 1 A.

Table D.2-8 - CISTPL_CFTABLE_ENTRY

Byte	Address $_{(hex)}$	7	6	5	4	3	2	1	0
0	00	TPL_CODE = CISTPL_CFTABLE_ENTRY (0x1B)							
1	02	TPL_LINK == 0x33							
2	04	TPCE_INDX = 0xC0 LOGICAL OR Config. Entry Number$_H$							
3	06	TPCE_IF = 0x04							
4	08	TPCE_FS = 0x0A							
5	0A	TPCE_PD Vcc Parameter Selection Byte = 0x38							
6	0C	TPCE_PD Vcc Static Current = Manufacturer value							
7	0E	TPCE_PD Vcc Average Current = 0x07							
8	10	TPCE_PD Vcc Peak Current = 0x07							
9	12	TPCE_PD Vpp Parameter Selection Byte = 0x78							
10	14	TPCE_PD Vpp Static Current = Manufacturer value							
11	16	TPCE_PD Vpp Average Current = 0x26							
12	18	TPCE_PD Vpp Peak Current = 0x26							

Byte	Address (hex)	7	6	5	4	3	2	1	0
13	1A	TPCE_PD Vpp 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)							
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$_{(hex)}$	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$_{(hex)}$	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 $_{(hex)}$	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$_{(hex)}$	7	6	5	4	3	2	1	0
0	00	SRESET	LevIREQ	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_INDX.

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 Previous Resource Versions and Associated APDUs

To ensure backwards compatibility when the Card and the Host negotiate a resource version and either the Host or the Card only supports the lower version of the resource, the other device is expected to use the lowest common version of that resource. The following section is a detail of the previous versions of the specific resources identified in Table 9.3-2 as well as their associated APDUs.

Table E-1 - Deprecated Resource Identifier Values

Resource	Class	Type	Version	Resource identifier
Application Information	2	1	1	0x00020041
Conditional Access Support	3	1	1	0x00030041
Host Control	32	1	1	0x00200041
Host Control	32	1	2	0x00200042
MMI	64	1	1	0x00400041
Homing	17	1	1	0x00110041
Low Speed Communication	96	50	3	0x00605043
Low Speed Communication	96	80	3	0x00608043
Copy Protection	176	1	1	0x00B00041
Copy Protection*	176	2	1	0x00B00102
Generic IPPV Support	128	1	1	0x00800041
System Control	43	1	2	0x002B0042
System Control	43	1	3	0x002B0043

E.1 Low Speed Communication Resource - Version 2

The Low Speed Communication resource, originally defined in [NRSSB], was modified to support the identification of the Forward Data Channel, the Reverse Data Channel and any type of Host's cable modem implementations. The modified Low Speed Communication resource was not a means for passing upstream/downstream OOB data to/from the Card via the CHI. All upstream/downstream OOB data SHALL be passed directly to/from the Card via the CHI, as defined in Section 5.3. Support of version 1 in [NRSSB] is optional.

Table E.1-1 - Low Speed Communication Resource (Version 2)

Resource	Mode	Class	Type	Version	Identifier (hex)
Low_Speed_Communication (Cable Return)	S-Mode/M-Mode	96	(*)	2	0x006xxx2

The Low_Speed_Communication Identifier can be any value between 0060002 and 0060FFF2. A Low Speed Communication resource instance is declared with a new specific identifier for each active Host communication device.

The Low_Speed_Communication resource type (*) is updated to describe different and multiple Cable return channels.

Table E.1-2 - Device Type Values

Device Type Field	Value
Telco modem	00-3F
Serial Ports	40-4F
Cable Return Channel	50-57
Reserved	60-7F
Host Modem (e.g. DOCSIS)	80-9F
Reserved	A0-FF

The 10-bit resource type field for the Cable Return Channel is coded into two fields—an 8-bit Device Type field and a 2-bit Device Number field.

Table E.1-3 - Cable Return Resource Type

Bit	9	8	7	6	5	4	3	2	1	0
	0	1	0	1	0	Channel type			Device no.	
	←	Device Type						→		
	←	Resource Type							→	

The Device Type field consists of a set of five hard-coded bits, as defined in the following table, and a Channel Type field. The Channel type field consists of three bits, each designating a separate channel as follows:

Channel Type	Bit 4-2
FDC and RDC	000
FDC Only	001
Reserved	010 - 111

FDC and RDC: Identifies that the Host is equipped with a Forward Data Channel (FDC) and a Reverse Data Channel (RDC).

FDC Only: Identifies that the Host is only equipped with a Forward Data Channel (FDC).

Low Speed Communication Resource Version 1 does not apply.

E.2 Copy Protection

E.2.1 Copy Protection - Type 2 Version 1 (Deprecated)

Table E.2-1 - Copy Protection Resource (Type 2 Version 1)

Resource	Mode	Class	Type	Version	Identifier (hex)
Copy Protection	S-Mode	176	2	1	0x00B00081

E.2.1.1 CP_open_req()

The Type 2 Version 1 ***CP_open_req()*** APDU is the same as defined in section 11.3.1.1 of [CCCP].

E.2.1.2 CP_open_cnf()

The Type 2 Version 1 ***CP_open_cnf()*** APDU is the same as defined in section 11.3.1.2 of [CCCP].

E.2.1.3 CP_data_req() Card's Authentication Data Message

This APDU object is issued by the Card to send its ID and random nonce to the Host to generate a new CP content key.

Table E.2-2 - Card's Authentication Data Message Syntax (Type 2 Version 1)

Message Syntax	bits	bytes	Description
CP_data_req () {			
CP_data_req_tag	24	3	Has the value: 0x9F 9002
Length_field()	8	1	length_field () is defined in CEA-679-C (Part B) section 7. Since there is no other field followed, length_field() shall have the following values set: size_indicator = 0, length_value = 27
CP_system_id	8	1	Has the value: 2 (CableCARD-CP System)
Send_datatype_nbr	8	1	Has the value: 2
For(i=0; i<Send_datatype_nbr; i++) {	(32)	(2*3)	
Datatype_ID	8	1	When i = 0, Datatype_ID value = 6 (POD_ID)
	8	1	When i = 1, Datatype_ID value = 12 (N_module)
Datatype_length	16	2	When i = 0, Datatype_length value = 8
	16	2	When i = 1, Datatype_length value = 8
For (j=0; j<Datatype_length; j++) {	(128)	(16)	
Data_type	64	8	When i = 0, Data_type = POD_ID (6) and
	64	8	When i = 1, Data_type = N_module
}			
}			
Request_datatype_nbr	8	1	Has the value: 2
For(i=0; i<Request_datatype_nbr; i++) {	(16)	(2*1)	
Datatype_ID	8	1	When i = 0, Datatype_ID value = 5 (Host_ID)
	8	1	When i = 1, Datatype_ID value = 11 (N_Host)
}			
}			

Table E.2-3 - CP_system_id Values

CP_system_id	ID Value (Binary)
No compatible CP system supported	XXX0 0000
System 1	XXX0 0001
System 2	XXX0 0010
Systems 3 to 30	XXX0 0011 to XXX1 1110
System 31	XXX1 1111
Message is Encrypted	1XXX XXXX
Message is Not Encrypted	0XXX XXXX

E.2.1.4 CP_data_cnf() Host's Authentication Data Message

This object also contains Host's ID and nonce so the Card can derive its CP content encryption key.

Table E.2-4 - Host's Authentication Data Message Syntax (Type 2 Version 1)

Message Syntax	bits	bytes	Description
CP_data_cnf () {			
CP_data_cnf_tag	24	3	Has the value: 0x9F 9003
Length_field()	8	3	length_field () is defined in CEA-679-C (Part B) section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 20
CP_system_id	8	1	Values are listed in CP_system_id Table above.
Send_datatype_nbr	8	1	Has the value: 2
For(i=0; i<Send_datatype_nbr; i++) {	(48)	(2*3)	
Datatype_ID	8	1	When i = 0, Datatype_ID = 5 (Host_ID); and
	8	1	When i = 1, Datatype_ID = 11 (N_Host)
Datatype_length	16	2	When i = 0, Datatype_length = 6
	16	2	When i = 1, Datatype_length = 8
For (j=0; j<Datatype_length; j++) {	(104)	(13)	
Data_type	40	5	When i = 0, Data_type = Host_ID (5)
	64	8	When i = 1, Data_type = N_Host;and
}			
}			
}			

E.2.1.5 CP_sync_req() and CP_sync_cnf()

The Type 2 Version 1 ***CP_sync_req()*** and ***CP_sync_cnf()*** APDU is the same as defined in section 11.6 of [CCCP].

E.2.1.6 CP_data_req() Card's Request for Host's AuthKey Message

The Type 2 Version 1 ***CP_datareq()Card's Request for Host's AuthKey Message*** APDU is the same as defined in section 11.4.2.1 of [CCCP].

E.2.1.7 CP_data_cnf() Reply Message with Host's AuthKey

This APDU object is issued by the Host to send its authentication key ($AuthKey_H$) to the Card.

Table E.2-5 - Host's Reply with AuthKey Message Syntax (Type 2 Version 1)

Message Syntax	bits	bytes	Description
CP_data_cnf () {			
CP_data_cnf_tag	24	3	Has the value: 0x9F 9003
length_field()	8	1	length_field () is defined in CEA-679-C (Part B) section 7. The length_field() in this message shall have the following values set: size_indicator = 0, length_value = 25
CP_system_id	8	1	Has the value: 2
Send_datatype_nbr	8	1	Has the value: 1
For(i=0; i<Send_datatype_nbr; i++) {	(16)	(2)	
Datatype_ID	8	1	Has the value: 22 ($AuthKey_H$,)
Datatype_length	16	2	Has the value: 20
For (j=0; j<Datatype_length; j++) {			
Data_type	160	20	Data_type = $AuthKey_H$
}			
}			
}			

E.2.1.8 CP_data_req() SATP Key Generation

The Type 2 Version 1 ***CP_data_req() SATO Key Generation*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.1.9 CP_data_cnf() CCI SATP Key Generation

The Type 2 Version 1 ***CP_data_cnf() CCI SATP Key Generation*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.1.10 CP_data_req() CCI SATP Transmission

Table E.2-6 - CD_data_req() CCI SATP Transmission (Type 2 Version 1)

Message Syntax	bits	bytes	Description
CP_data_req(){			
CP_data_req_tag	24	3	Has the value: 0x9F 9002
length_field()	8	1	Has the value of 0x23 size_indicator = 0, length_value = 35
CP_system_id	8	1	Has the value of 2
Send_datatype_nbr	8	1	Has the value of 3
for(i=0; i<Send_datatype_nbr; i++)			
{ Datatype_id	8	1	For i = 0, Datatype_id = 25 (CCI_data)
	8	1	For i = 1, Datatype_id = 26 (program_number)
	8	1	For i = 2, Datatype_id = 27 (CCI_auth)
Datatype_length	16	2	For i = 0, Datatype_length = 0x0001
	16	2	For i = 1, Datatype_length = 0x0002
	16	2	For i = 2, Datatype_length = 0x0014
for (j=0; j<Datatype_length; j++)			
{ Data_type	8	1	For i = 0, Data_type = CCI_data.
	16	2	For i = 1, Data_type = program_number.
	160	20	For i = 2, Data_type = CCI_auth.
}			
}			
Request_datatype_nbr	8	1	Has the value of 2
for(i=0; i<Request_datatype_nbr; i++)			
{ Datatype_id	8	1	For i=0, Datatype_id = 28 (CCI_ack)
	8	1	For i=1, Datatype_id = 26 (program_number)
}			
}			

E.2.1.11 CP_data_cnf() CCI SATP Transmission

The Type 2 Version 1 ***CP_data_cnf() CCI Transmission*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.2 Copy Protection Type 4 Version 1

Table E.2-7 - Copy Protection Resource (Type 4 Version 1)

Resource	Mode	Class	Type	Version	Identifier (hex)
Copy Protection	M-Mode	176	4	1	0x00B00101

E.2.3 CP_open_req()

The Type 4 Version 1 ***CP_open_req()*** APDU is the same as defined in section 11.3.1.1 of [CCCP].

E.2.4 CP_open_cnf()

The Type 4 Version 1 ***CP_open_cnf()*** APDU is the same as defined in section 11.3.1.2 of [CCCP].

E.2.5 CP_data_req() Card's Authentication Data Message

The Type 4 Version 1 ***CP_data_req()*** APDU is the same as defined in section 11.4.1.1 of [CCCP].

E.2.6 CP_data_cnf() Host's Authentication Data Message

The Type 4 Version 1 ***CP_data_cnf()*** APDU is the same as defined in section 11.4.1.2 of [CCCP].

E.2.7 CP_data_req() Card's Request for Auth Key

The Type 4 Version 1 ***CP_data_req()*** APDU is the same as defined in section 11.4.2.1 of [CCCP].

E.2.8 CP_data_cnf() Reply Message with Host's AuthKey

The Type 4 Version 1 ***CP_data_cnf()*** APDU is the same as defined in section 11.4.2.2 of [CCCP].

E.2.9 CP_data_req() Card's CPKey Generation Message

The Type 4 Version 1 ***CP_data_req()*** APDU is the same as defined in section 11.5 of [CCCP].

E.2.10 CP_data_cnf() Host's CPKey Generation Message

The Type 4 Version 1 ***CP_data_cnf()*** APDU is the same as defined in section 11.5 of [CCCP].

E.2.11 CP_sync_req() Card's CPKey Ready Message

The Type 4 Version 1 ***CP_sync_req()*** APDU is the same as defined in section 11.6 of [CCCP].

E.2.12 CP_sync_cnf() Host's CPKey Ready Message

The Type 4 Version 1 ***CP_sync_cnf()*** APDU is the same as defined in section 11.6 of [CCCP].

E.2.13 CP_data_req() Card's CCI Challenge Message

The Type 4 Version 1 ***CP_data_req()*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.14 CP_data_cnf() Host's CCI Response Message

The Type 4 Version 1 ***CP_data_cnf()*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.15 CP_data_req() CCI Delivery Message

The Type 4 Version 1 ***CP_data_req()*** APDU is the same as defined in section 11.7 of [CCCP].

E.2.16 CP_data_cnf() CCI Acknowledgement Message

The Type 4 Version 1 ***CP_data_cnf()*** APDU is the same as defined in section 11.7 of [CCCP].

E.3 Specific Application Support - Type 1 Version 1

Table E.3-1 - Specific Application Support Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Specific Application Support	S-Mode	144	1	1	0x00900041

E.3.1 SAS_connect_reqst()

The Type 1 Version 1 ***SAS_connect_rqst()*** APDU is the same as defined in Section 9.17.1.

E.3.2 SAS_connect_cnf()

The Type 1 Version 1 ***SAS_connect_cnf()*** APDU is the same as defined in Section 9.17.2.

E.3.3 SAS_data_reqst()

The Type 1 Version 1 ***SAS_data_reqst()*** APDU is the same as defined in Section 9.17.3.

E.3.4 SAS_data_av()

The Type 1 Version 1 ***SAS_data_av()*** APDU is the same as defined in Section 9.17.4.

E.3.5 SAS_data_av_cnf()

The Type 1 Version 1 ***SAS_data_av_cnf()*** APDU is the same as defined in Section 9.17.5.

E.3.6 SAS server_query()

The Type 1 Version 1 ***SAS_server_query()*** APDU is the same as defined in Section 9.17.6.

E.3.7 SAS_server_reply()

The Type 1 Version 1 ***SAS_server_replyquery()*** APDU is the same as defined in Section 9.17.7.

E.4 Generic IPPV Support - Type 2 Version 1 (Deprecated)

Table E.4-1 -Generic IPPV Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic IPPV Support	S-Mode	128	2	1	0x00800081

This resource includes the following objects:

Table E.4-2 - Generic IPPV Support

APDU Name	Tag Value	Resource	Direction Host ↔ CableCARD
Program_req()	0x9F8F00	Generic IPPV Support	→
Program_cnf()	0x9F8F01	Generic IPPV Support	←
Purchase_req()	0x9F8F02	Generic IPPV Support	→
Purchase_cnf()	0x9F8F03	Generic IPPV Support	←
Cancel_req()	0x9F8F04	Generic IPPV Support	→
Cancel_cnf()	0x9F8F05	Generic IPPV Support	←
History_req()	0x9F8F06	Generic IPPV Support	→
History_cnf()	0x9F8F07	Generic IPPV Support	←

E.4.1 Program_req() & Program_cnf()

The Host's navigation application SHALL use the ***Program_req()*** object to request the Card's CA information on a particular program.

The Card SHALL respond with the ***Program_cnf()*** object to the ***Program_req()*** request.

Table E.4-3 - Program Request Object Syntax

Syntax	No. of bits	Mnemonic
program_req() {		
program_req_tag	24	uimsbf
length_field()		
transaction_id	8	uimsbf
transport_stream_id	16	uimsbf
program_number	16	uimsbf
source id	16	uimsbf
event_id	16	uimsbf
current_next indicator	8	uimsbf
reserved	7	
current_next	1	uimsbf
program_info_length	8	
for (i=0; i < program_info_length; i++) {		
ca_descriptor()		
/* ca descriptor at program level*/		
}		
}		

program_req_tag 0x9F8F00

transaction_id This field is a unique number generated by the Host to uniquely identify this transaction. The associated ***program_cnf()*** message will include this *transaction_ID* value. Hosts SHALL maintain a *transaction_ID* counter and increment it by 1 (mod 256) for each new transaction.

transport_stream_id A 16-bit unsigned integer field, in the range 0x0000 to 0xFFFF, that represents the MPEG-2 Transport Stream ID associated with the program being requested.

program_number A 16-bit unsigned integer number indicating the program that is being requested.

source_id A 16-bit unsigned integer number indicating the source_id of the program that is being requested. (This text should be inserted after the program_number field description.)

event_id A 16-bit unsigned integer number specifying the event requested on the specified program_number. If the Event_ID is unknown, this field SHALL be set to all 0s.

current_next Used to specify the current or next event on the specified program_number. Only relevant when Event_ID is set to 0. When not set, indicates that the current event is being requested. When set, indicates that the next event is requested.

program_info_length These fields SHALL be used by the Host to provide the Card with every program level *ca_ descriptor* of this MPEG program.

ca_descriptor The CA descriptor SHALL be extracted from the PMT table by the Host navigation application.

Table E.4-4 - Program Confirm Object Syntax

Syntax	No. of bits	Mnemonic
program_cnf() {		
program_cnf_tag	24	uimsbf
length_field()		
transaction_id	8	uimsbf
status_field	8	uimsbf
if (status_field == 0) {		
option_nb	8	uimsbf
for (option_id=1; I <= option_nb;		
option_id++) {		
purchase_type	8	uimsbf
purchase_price	16	uimsbf
purchase_validation	8	uimsbf
expiration_date	32	uimsbf
program_start_time	32	uimsbf
initial_Free_preview_duration	16	uimsbf
anytime_free_preview_duration	16	uimsbf
title_length	8	uimsbf
for (J=0; J < title_length; J++) {		
title_txt	8	uimsbf
}		
text_length	8	uimsbf
for (J=0; J < text_length; J++) {		
text_txt	8	uimsbf
}		
descriptor_length	16	uimsbf
for (k=0; k < desc_length; k++) {		
descriptor()	var	uimsbf
}		
}		
}		
}		

program_cnf_tag 0x9F8F01

transaction_id This field is the transaction_id number sent to the Card from the Host in the transaction_id field from the ***program_request()***.

status_field This field returns the status of the ***program_req()***. If the Card can provide the requested information on the pointed event, then Status_field SHALL be set to 0x00. Otherwise it will be set to one of the following values.

0x00 Request Granted
0x01 Request Denied - Card busy
0x02 Request Denied - Unknown Event
0x03-0xFF Reserved

option_nb This field defines the number of options under which a particular event can be purchased.

purchase_type This field characterizes how the event may be purchased.

0x00 Viewing Only
0x01 Viewing and Right to Copy Once
0x02 Viewing and Right to Copy Unlimited
0x03 Subscription
0x04 Purchased for Viewing Only

0x05 Purchased with Viewing and Right to Copy Once
0x06 Purchased with Viewing and Right to Copy Unlimited
0x07 Un-Purchasable
0x08-0xFF Reserved

Viewing only This program may be purchased for viewing only, without the right to make any copies, as defined by the operator.
Note: Through private agreements between a cable operator and content providers, the cable operator determines the pricing and right to copy options appropriate for its market.

Viewing and Right to Tape Copy Once This program may be purchased for viewing and with the right to copy the analog video output and make one copy as defined by the operator.

Viewing and Right to Copy Unlimited This program may be purchased for viewing and with the right to make unlimited copies as defined by the operator.

Subscription This program is a subscription event, and is not purchasable as an IPPV event.

Purchased for Viewing Only This program has already been purchased with viewing rights only, and without the right to make any copies as defined by the operator.

Purchased with Viewing and Right to Tape Copy Once This program has already been purchased for viewing with the right to tape and right to make one copy as defined by the operator.

Purchased with Viewing and Right to Copy Unlimited This program has already been purchased for viewing with the right to make unlimited copies as defined by the operator.

Un-purchasable This is not a purchasable program.

Reserved These values are reserved.

purchase_price This 2-byte field provides event pricing information. The event price is given by the Denomination unit multiplied by the Value. For example, if the Denomination unit is 5 cents, and the Value is 79, the price would be $3.95. The format is further defined in Table E.4-5.

Table E.4-5 - Purchase Price for Program Confirm

Bit	7	6	5	4	3	2	1	0
	Denomination unit in cents (MS) Value (LS)							

purchase_validation This parameter defines the level of validation the Card expects to validate the purchase. The values are as follows:

0x00 No CA validation required
0x01 PIN code required for Purchase transaction
0x02 PIN code required for Cancel transaction
0x03 PIN code required for History transaction
0x04 PIN code required for Purchase and Cancel transactions
0x05 PIN code required for Purchase and History transaction
0x06 PIN code required for Purchase, Cancel, History transactions
0x07-0xFF Reserved

expiration_date This field contains the expiration time of the event. It is a 32-bit unsigned integer quantity representing the expiration time as the number of seconds since 12 AM, January 6, 1980.

program_start_time A 32-bit unsigned integer, defining the start time of the program, in GPS seconds since 12 AM January 6, 1980.

initial_free_preview_duration A 16-bit unsigned integer, defining the duration of the free preview period. The duration is measured from the program_start_time.

anytime_free_preview duration A 16-bit unsigned integer, defining the duration of the Anytime_free_preview.

title_length, title_txt These fields allow the Card to provide a purchase option title.

text_length_txt These fields allow the Card to provide a purchase option text.

desc_length A 16-bit unsigned integer that indicates the length of the block of optional descriptors to follow. If no descriptors are present, the length SHALL indicate zero.

descriptor() A data structure of the form type-length-data, where *type* is an 8-bit descriptor type identifier, *length* is an 8-bit field indicating the number of bytes to follow in the descriptor, and *data* is arbitrary data. The syntax and semantics of the data are as defined for the particular type of descriptor. The *content_advisory_descriptor()* (as defined in section 6.7.4 of ATSC A/65) may be used to indicate the rating of the program. The program rating SHALL be coded according to the MPAA and V-Chip Rating and Content Advisories to be used for parental restrictions on program purchases.

E.4.2 Purchase_req() & Purchase_cnf()

The Host's navigation application SHALL use the ***purchase_req()*** object to request a purchase of a particular program offer.

The Card SHALL respond with the ***purchase_cnf()*** object to the *purchase_req()* request.

Table E.4-6 - Purchase Request Object Syntax

Syntax	No. of bits	Mnemonic
`purchase_req() {`		
`  purchase_req_tag`	24	uimsbf
`  length_field()`		
`  transaction_id`	8	uimsbf
`  option_id`	8	uimsbf
`  PINcode_length`	8	uimsbf
`  for (I=0; I<=PINcode_length; I++) {`		
`  PINcode_byte`	8	uimsbf
`  }`		
`}`		

purchase_req_tag 0x9F8F02

transaction_id A number supplied by the Host issued from an 8-bit cyclic counter that identifies each ***purchase_req()*** APDU and allows the Host to identify each ***purchase_cnf()*** received from the Card.

option_id The possible ways to purchase an event, up to the maximum value.

PINcode_length, PINcode_byte These fields allow the Host navigation application to pass the requested PIN code to the Card. In case no PIN code was requested, the *PINcode_length* is set to '0'.

Table E.4-7 - Purchase Confirm Object Syntax

Syntax	No. of bits	Mnemonic
p_cnf() {		
purchase_cnf_tag	24	uimsbf
length_field()		
transaction_id	8	uimsbf
option_id	8	uimsbf
status_field	8	uimsbf
IPPVslot_id	8	uimsbf
status_register	8	uimsbf
comment_length	8	uimsbf
for (I=0; I<= comment_length; I++) {		
comment_txt	8	uimsbf
}		
}		

purchase_cnf_tag 0x9F8F03

transaction_id A number supplied by the Host issued from an 8-bit cyclic counter that identifies each ***purchase_req()*** APDU and allows the Host to identify each ***purchase_cnf()*** received from the Card.

option_id The possible ways to purchase an event, up to the maximum value.

status_field This field returns the status of the ***purchase_req().*** If the Card has validated the purchase, then *status_field* shall be set to 0x00. Otherwise it will be set to one of the following values. When there is more than one reason to deny the purchase, *status_field* is set to the lowest applicable value. These values are as follows:

0x00 Purchase Granted
0x01 Purchase Denied - Card busy
0x02 Purchase Denied - Unknown Transaction ID or Option ID
0x03 Purchase Denied - Invalid PIN code
0x04 Purchase Denied - Event already purchased
0x05 Purchase Denied - Blackout is active
0x06 Purchase Denied - Credit Limit exceeded
0x07 Purchase Denied - IPPV Slot Limit is exceeded
0x08 Purchase Denied - Spending Limit is exceeded
0x09 Purchase Denied - Rating Limit is exceeded
0x0A Purchase Denied - Check Comments
0x0B-0xFF Reserved

Purchase Denied IPPV_slot_limit is exceeded The Card is unable to make additional IPPV purchases until it has reported all of its unreported purchases to the headend.

IPPVslot_id If *status_field* is 0x00 (Purchase Granted) then *IPPVslot_id* will contain the unique slot identifier that will later identify the purchasing transaction. If *status_field* is any other value, *IPPVslot_ID* is reset to 0.

Comment_length, Comment_txt These fields allow the Card to explain, using plain text, why the purchase request has been granted or denied.

Status_register This field identifies the CA status of the program event. The designation of each bit is summarized in the following table.

Table E.4-8 - Status Register for Purchase Confirm

Bit	7	6	5	4	3	2	1	0
	VPU	OPU	UPU	AUT	FRE	REP	CAN	VIE

VPU is set to 1 when the program event has been purchased for viewing once.

OPU is set to 1 when the program event has been purchased for taping once.

UPU is set to 1 when the program event has been purchased for unlimited taping.

AUT is set to 1 when the program event has been authorized.

FRE is set to 1 when the free preview (initial or anytime) of the program event has been viewed.

REP is set to 1 when the program event has been reported.

CAN is set to 1 when the program event has been cancelled.

VIE is set to 1 when the program event has been viewed.

E.4.3 Cancel_req() & Cancel_cnf()

The Host's navigation application SHALL use the ***cancel_req()*** object to request a cancellation of a particular purchased program offer.

The Card SHALL respond with the ***cancel_cnf()*** object to the ***cancel_req()*** request.

Table E.4-9 - Cancel Request Object Syntax

Syntax	No. of bits	Mnemonic
`cancel_req() {`		
`  cancel_req_tag`	`24`	`uimsbf`
`  length_field()`		
`  IPPVslot_ID`	`8`	`uimsbf`
`  PINcode_length`	`8`	`uimsbf`
`  for (I=0; I<=PINcode_length; I++) {`		
`    PINcode_byte`	`8`	`uimsbf`
`  }`		
`}`		

cancel_req_tag 0x9F8F04

IPPVslot_id If *status_field* is 0x00 (Purchase Granted) then *IPPVslot_id* will contain the unique slot identifier that will later identify the purchasing transaction. If *status_field* is any other value, *IPPVslot_ID* is reset to 0.

PINcode_length, PINcode_byte These fields allow the Host navigation application to pass the requested PIN code to the Card. In case no PIN code was requested, the *PINcode_length* is set to '0'.

Table E.4-10 - Cancel Confirm Object Syntax

Syntax	No. of bits	Mnemonic
cancel_cnf() {		
cancel_cnf_tag	24	uimsbf
length_field()		
IPPVslot_id	8	uimsbf
status_field	8	uimsbf
status_register	8	uimsbf
comment_length	8	uimsbf
for (I=0; I<= comment_length; I++) {		
comment_txt	8	uimsbf
}		
}		

cancel_cnf_tag 0x9F8F05

IPPVslot_id If *status_field* is 0x00 (Purchase Granted) then *IPPVslot_id* will contain the unique slot identifier that will later identify the purchasing transaction. If *status_field* is any other value, *IPPVslot_ID* is reset to 0.

status_field This field returns the status of the ***cancel_req().*** If the Card has validated the cancellation, then *status_field* SHALL be set to 0x00. Otherwise it will be set to one of the following values. When there is more than one reason to deny the cancellation, *status_field* is set to the lowest applicable value. These values are defined as follows:

0x00 Cancellation Granted
0x01 Cancellation Denied - Card busy
0x02 Cancellation Denied - Unknown IPPV slot id
0x03 Cancellation Denied - Invalid PIN code
0x04 Cancellation Denied - Program already viewed or in progress
0x05-0x09 Reserved
0x0A Cancellation Denied - Check Comments
0x0B-0xFF Reserved

E.4.4 History_req() & History_cnf()

The Host's navigation application SHALL use the ***history_req()*** object to request the history of all purchased and cancelled program events held in the Card's memory.

The Card SHALL respond with the ***history_cnf()*** object to the ***history_req()*** request.

Table E.4-11 - History Request Object Syntax

Syntax	No. of bits	Mnemonic
history_req() {		
history_req_tag	24	uimsbf
length_field()		uimsbf
PINcode_length	8	uimsbf
for (I=0; I<=PINcode_length; I++) {		
PINcode_byte	8	uimsbf
}		
}		

history_req_tag 0x9F8F06

PINcode_length, PINcode_byte These fields allow the Host navigation application to pass the requested PIN code to get IPPV history on events that required a PIN Code validation for History. In case no PIN code or a wrong PIN code is supplied, only history on events that do not require PIN Code validation for History will be provided.

Table E.4-12 - History Confirm Object Syntax

Syntax	No. of bits	Mnemonic
history_cnf() {		
history_cnf_tag	24	uimsbf
length_field()		
status_field	8	uimsbf
comment_length	8	uimsbf
for (i=0; i<= comment_length; i++) {		
comment_txt	8	uimsbf
}		
ippvslot_nb	8	uimsbf
for (i=0; i<= ippvslot_nb; i++) {		
ippvslot_id	8	uimsbf
purchase_type	8	uimsbf
purchase_price	16	uimsbf
status_register	8	uimsbf
purchase_date	32	uimsbf
cancel_date	32	uimsbf
event_date	32	uimsbf
title_length	8	uimsbf
for (j=0; j < title_length; j++) {		
title_txt	8	uimsbf
}		
text_length	8	uimsbf
for (j=0; j < text_length; j++) {		
text_txt	8	uimsbf
}		
descriptor_length	16	uimsbf
for (k=0; k < desc_length; k++) {		
descriptor()	var	
}		
}		
}		

history_cnf_tag 0x9F8F07

Status_field This field returns the status of the ***history_req().*** If the Card has validated the History request, then *status_field* SHALL be set to 0x00. Otherwise it will be set to one of the following values.

0x00 History Granted
0x01 History Denied - Card busy
0x02 Reserved
0x03 History Denied - Invalid PIN code
0x04-0x09 Reserved
0x0A History Denied - Check Comments
0x0B-0xFF Reserved

Purchase_date, Cancel_date, Event_date These fields contain respectively the purchase time, the cancel time and the starting time of the event. They are 32-bit unsigned integer quantities representing the time as the number of seconds since 12 AM, January 6, 1980.

If the *Cancel_date* field contains all FFFFs, this indicates that no appropriate value is available for this field.

E.5 Generic Diagnostics Type 1 Version 1

Table E.5-1 - Generic Diagnostics Support Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Generic Diagnostic Support	S-Mode	260	1	1	0x01040041

The following values SHALL be used as the diagnostic_id for Type 1 Version 1 of the Generic Diagnostic Support resource.

Table E.5-2 - Diagnostic Ids

Diagnostic	Value
Set-top 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
Reserved	0x08 - FF

Table E.5-3 - diagnostic_cnf APDU Syntax (Type 1, Version 1)

Syntax	No. of Bits	Mnemonic
diagnostic_cnf() {		
diagnostic_cnf_tag	24	uimsbf
length_field()		
number_of_diag	8	uimsbf
for (i=o; 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()		
}		
}		
}		

For field descriptions of ***diagnostic_cnf()*** APDU, see Section 9.16.2.

E.5.1 memory_report

Memory reports SHALL contain the memory parameters associated with the Host.

Table E.5-4 - memory_report (Type 1 Version 1)

Syntax	No. of Bits	Mnemonic
memory_report() {		
number_of_memory	8	uimsbf
if (i=0; i<number_of_memory; i++) {		
memory_type	8	uimsbf
memory_size	32	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 Internal Hard drive, no DRM (Digital Rights Management) support
0x06 Video memory
0x07 Other memory
0x08 - FF Reserved

memory_size Designates the physical size of the specified memory type. The units are kilobytes, defined to be 1,024 bytes.

E.5.2 software_ver_report

The Type 1 Version 1 ***software_ver_report()*** APDU is the same as defined in Section 9.16.3.2.

E.5.3 firmware_ver_report

The Type 1 Version 1 ***firmware_ver_report()*** APDU is the same as defined in Section 9.16.3.3.

E.5.4 MAC_address_report

The MAC address report SHALL contain the MAC address parameters associated with the Host.

Table E.5-5 - MAC_address_report (Type 1 Version 1)

Syntax	No. of Bits	Mnemonic
MAC_address_report() {		
number_of_addresses	8	uimsbf
for (i=0; i<number_of_addresses; i++) {		
MAC_address_type	8	uimsbf
number_of_bytes	8	uimsbf
for (j=0; j<number_of_bytes; j++) {		
MAC_address_byte	8	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 USB
0x04 DOCSIS
0x05 Ethernet
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 two hexadecimal values (xx) in the range of 0x00 to 0xFF.

E.5.5 FAT_status_report

The Type 1 Version 1 ***FAT_status_report()*** APDU is the same as defined in section 9.16.3.5.

E.5.6 FDC_Status_report

In response to a FDC status report request, the Host SHALL reply with an FDC status report, unless an error has occurred.

Table E.5-6 - FDC_status_report (Type 1 Version 1)

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 E.5-7 - 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)							

carrier_lock_status Indicates if the current carrier is locked or not locked.

0b Not locked
1b Locked

packet_sync_status Indicates if the current FDC packets are in sync

0b Not in sync
1b In sync

E.5.7 current_channel_report

The Type 1 Version 1 ***current_channel_report()*** APDU is the same as defined in Section 9.16.3.7.

E.5.8 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 E.5-8 - 1394_port_report (Type 1 Version 1)

Syntax	No. of Bits	Mnemonic
1394_port_report() {		
reserved	3	'111'
loop_status	1	bslbf
root_status	1	bslbf
cycle_master_status	1	bslbf
port_1_connection_status	1	bslbf
port_2_connection_status	1	bslbf
total_number_of_nodes	16	uimsbf
}		

loop_status Indicates if a loop exists on the 1394 bus.

0b No loop exists
1b Loop exists

root_status Indicates if the Host device is the root node on the 1394 bus.

0b Not root
1b Is root

cycle_master_status Indicates if the Host device is the cycle master node on the 1394 bus.

0b Not cycle master
1b Is cycle master

port_1_connection_status Indicates if port 1 of the 1394 PHY is connected to a 1394 bus.

0b Not connected
1b Connected

port_2_connection_status Indicates if port 2 of the 1394 PHY is connected to a 1394 bus.

0b Not connected
1b Connected

total_number_of_nodes 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).

E.6 System Control

Table E.6-1 - System Control Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
System Control	S-Mode/ M-Mode	43	1	1	0x002B0041
System Control	S-Mode/ M-Mode	43	1	2	0x002B0042
System Control	S-Mode/ M-Mode	43	1	3	0x002B0043

E.6.1 host_info_request()

Table E.6-2 - host_info_request (Type 1 Version 1)

Syntax	No. of bits	Mnemonic
host_info_request() {		
host_info_request_tag	24	uimsbf
length_field()		
supported_download_type	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

The Type 1 Version 2 and Type 1 Version 3 of the ***host_info_request()*** APDU is the same as defined in section 6.2.1 of [CDL].

E.6.2 host_info_response()

The Type 1 Version 1, Type 1 Version 2 and Type 1 Version 3 of the ***host_info_response()*** APDU is the same as defined in section 6.2.2 of [CDL].

E.6.3 code_version_table()

The Type 1 Version 1 ***code_version_table()*** APDU is the same as defined in section 6.2.3 of [CDL].

Table E.6-3 - code version table (Type 1 Version 2)

Syntax	No. of bits	Mnemonic
code_version_table() {		
code_version_table_tag	24	uimsbf
length_field()		
configuration_count_change	8	uimsbf
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
if (download_type == 00) {		
frequency_vector	16	uimsbf
modulation_type	8	uimsbf
reserved	3	uimsbf
PID	13	uimsbf
}		

Syntax	No. of bits	Mnemonic
if (download_type == 01) {		
DSG_BT_address	48	uimsbf
source_ip_address	64	uimsbf
destination_ip_address	64	uimsbf
source_port_number	16	uimsbf
destination_port_number	16	uimsbf
application_id	16	uimsbf
PID	13	uimsbf
}		
if (download_type == 02) {		
tftp_server_address	64	uimsbf
)		
code_file_name_length	8	uimsbf
for(i=0;i<software_filename_length;i++){		
code_file_name_byte	8	uimsbf
}		
code_verification_certificate()		
}		

code_version_table_tag Value = 0x9F9C02

configuration_count_change Incremented by one (modulo the field size) by the Headend whenever any of the values of the Code Version Table for a given Host, defined by combination of the OUI and hardware_version_id or Host MAC address or Host ID, has been changed.

Note (Informative): In some events (for example, a failover or hot swap at the headend) discontinuities in the value of configuration change count may occur. After any event that can cause a discontinuity in the configuration change count, the Headend MUST ensure that the configuration change count is incremented (modulo the field size) between two subsequent CVT messages (even if the CVT message does not change). This is done to ensure that, after a failover or hot swap in the headend, the new configuration change count does not match the configuration change count used before the failover event. When the configuration change count is changed, the Card SHALL pass a CVT to the Host for verification if download is required.

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 = 24). Unique Identifier (the vendor's OUID) assigned to each vendor. Host sends the vendor ID to the Card to allow the Card to filter the CVT. A value of 0x0000 is not valid.
- 1 descriptor_data is hardware_version_id (mandatory, descriptor_len = 32), Unique Hardware identifier assigned to each type of hardware from a particular vendor. Host sends the hardware version ID to the Card to allow the Card to filter the CVT. This can be transmitted to the Headend by the Card. A value of 0x0000 SHALL NOT be permitted.

2 host_MAC_addr (optional, descriptor_len = 48), Host MAC address used for optional filtering if non-zero.
3 host_ID (optional, descriptor_len = 40), Host device's unique identification number used for optional filtering if non-zero or this parameter is present.
4-127 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.

download _type Way of delivery of the DSM-CC data carousel (supplied by Headend):

0x00 In-Band FAT Channel
0x01 DSG Channel
0x02 DOCSIS tftp

download_command When to download (supplied by Headend):

0x00 Download Now - If the vendor_id and hardware_version_id and optionally either a host_MAC_addr or host_id in the descriptor_data in the CVT matches that of the Host and the code_file_name in the CVT does not match that of the Host, then the download SHALL be initiated. If there is a match of the vendor_id and hardware_version_id, but the code_file_name in the CVT matches that of the Host, then the download SHALL NOT be initiated.
0x01 Deferred Download - The initiation of the download SHALL be deferred according to policies set in an OCAP Monitor Application. In the event that a Monitor Application is not available or no policies have been set, the Download Now scenario SHALL apply.
0x02-03 reserved

frequency_vector Frequency of the download carousel. The frequency is coded as the number of 0.25 MHz intervals.

modulation_type Possible Values:

0x00 Reserved
0x01 FAT Channel/QAM64
0x02 FAT Channel/QAM256
0x03 - 0xFF Reserved

PID Stream identifier of the code file.

tftp_server_address The IP address of the TFTP server where the code image resides. The code file name, as defined via the code_file_name_byte field, contains the complete directory path and name of the file to download. The address is 64 bits in length to support IPv6.

DSG_BT_address MAC address of the DSG broadcast tunnel.

source_ip_address The source IP address associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the source IP address of the packet and provide an additional filter at either the destination IP address (if defined) or a layer below the IP layer (e.g., Port and/or MPEG section filtering).

destination_ip_address The destination IP address associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the destination IP address of the packet and provide an additional filter at either the source IP address (if defined) or a layer below the IP layer (e.g., Port and/or MPEG section filtering).

source_port_number	The UDP source port number associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the source UDP header and provide an additional filter at a layer below the UDP layer (e.g., MPEG section filtering).
destination_port_number	The UDP destination port number associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the UDP header and provide an additional filter at a layer below the UDP layer (e.g., MPEG section filtering).
application_id	A DSG data stream identifier, associated with Type 4 Broadcast Tunnel information. The application_id value SHALL be greater than 0. Applicable only for the Host operating in DSG Advanced mode. This is utilized to allow the Host to build a DSG data stream routing table. This application ID SHALL be associated with the filter setting, for a Type 4 Broadcast Tunnel, passed to the Host in the ***configure_advanced_DSG()*** APDU.
PID	A DSG data stream identifier, associated with DSG Tunnel information. Applicable only for the Host operating in DSG Basic mode. This is utilized to allow the Host to open the MPEG section service_type data flow, from the Card to the Host, to receive an MPEG sections with code object delivered to the Card from DSG Broadcast Tunnel.
code_file_name_length	Length of code file name.
code_file_name_byte	Name of software upgrade file on carousel. This is the name of the Code File (see [SCTE23-2]) 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 ***code_version_table()*** 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.
code_verification_certificate	Authentication certificate(s) per [SCTE23-2].

Table E.6-4 - code version table (Type 1 Version 3)

Syntax	No. of bits	Mnemonic
code_version_table() {		
code_version_table_tag	24	uimsbf
length_field()		
configuration_count_change	8	uimsbf
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
if (download_type == 00) {		
location_type	8	uimsbf
if (location_type == 0) {		
source_ID	16	uimsbf
}		
if (location_type == 1) {		
frequency_vector	16	uimsbf
modulation_type	8	uimsbf
reserved	3	uimsbf
PID	13	uimsbf
}		
if (location_type == 2) {		
frequency_vector	16	uimsbf
modulation_type	8	uimsbf
program_number	16	uimsbf
}		
}		
if (download_type == 01) {		
DSG_BT_address	48	uimsbf
source_ip_address	128	uimsbf
destination_ip_address	128	uimsbf
source_port_number	16	uimsbf
destination_port_number	16	uimsbf
application_id	16	uimsbf
reserved	3	uimsbf
PID	13	uimsbf
}		
if (download_type == 02) {		
tftp_server_address	128	uimsbf
)		
code_file_name_length	8	uimsbf
for(i=0;i<software_filename_length;i++){		
code_file_name_byte	8	uimsbf
}		
number_of_cv_certificates	8	uimsbf
for(i=0;i<number_of_cv_certificates;++){		
certificate_type	8	uimsbf
code_verification_certificate()		
}		
}		

code_version_table_tag Value = 0x9F9C02

configuration_count_change

Incremented by one (modulo the field size) by the Headend whenever any of the values of the Code Version Table for a given Host, defined by combination of the OUI and hardware_version_id, file name Host MAC address or Host ID, has been changed.

Note (Informative): In some events (for example a failover or hot swap at the headend) discontinuities in the value of configuration change count may occur. After any event that can cause a discontinuity in the configuration change count, the Headend MUST ensure that the configuration change count is incremented (modulo the field size) between two subsequent CVT messages (even if the CVT message does not change). This is done to ensure that, after a failover or hot swap in the headend, the new configuration change count does not match the configuration change count used before the failover event. When the configuration change count is changed, the Card SHALL pass a CVT to the Host for verification if download is required.

number_of_descriptors

SHALL be greater than two; mandatory descriptors are vendor_id and hardware_version_id.

descriptor_tag

Possible Values:

- 0 descriptor_data is vendor_id (mandatory, descriptor_len = 24). Unique Identifier (the vendor's OUID) assigned to each vendor. Host sends the vendor ID to the Card to allow the Card to filter the CVT. A value of 0x0000 is not valid.
- 1 descriptor_data is hardware_version_id (mandatory, descriptor_len = 32), Unique Hardware identifier assigned to each type of hardware from a particular vendor. Host sends the hardware version ID to the Card to allow the Card to filter the CVT. This can be transmitted to the Headend by the Card. A value of 0x0000 SHALL NOT be permitted.
- 2 host_MAC_addr (optional, descriptor_len = 48), Host MAC address used for optional filtering if non-zero.
- 3 host_ID (optional, descriptor_len = 40), Host device's unique identification number used for optional filtering if non-zero or this parameter is present.
- 4-127 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.

download _type

Code file delivery method (the DSM-CC data carousel will be used for download_type 00 and download_type 01):

- 0x00 In-Band FAT Channel
- 0x01 DSG Channel
- 0x02 DOCSIS tftp

download_command

When to download (supplied by Headend):

- 0x00 Download Now - If the vendor_id and hardware_version_id and optionally either a host_MAC_addr or host_id in the descriptor_data in the CVT matches that of the Host and the code_file_name in the CVT does not match that of the Host, then the download SHALL be initiated. If there is a match of the vendor_id and hardware_version_id, but the code_file_name in the CVT matches that of the Host, then the download SHALL NOT be initiated.
- 0x01 Deferred Download - The initiation of the download SHALL be deferred according to policies set in an OCAP Monitor Application. In

the event that a Monitor Application is not available or no policies have been set, the Download Now scenario SHALL apply.

0x02-03 reserved

location_type Determines nature of the locator for DSM-CC data carousel carrying code file.

0x00 Carousel located by source_id
0x01 Carousel located by frequency vector and PID.
0x02 Carousel located by frequency and program number.
0x03-0xFF Reserved

source_ID The VCT source ID that is associated with each program source. The source ID is utilized to locate the frequency on which the DSM-CC data carousel is multiplexed.

frequency_vector Frequency of the download carousel. The frequency is coded as the number of 0.25 MHz intervals.

modulation_type Possible Values:

0x00 Reserved
0x01 FAT Channel/QAM64
0x02 FAT Channel/QAM256
0x03 - 0xFF Reserved

PID Stream identifier of the code file.

program_number Defines the program number in the transport stream that identifies the DSM-CC data carousel.

DSG_BT_address MAC address of the DSG broadcast tunnel.

source_ip_address The source IP address associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the source IP address of the packet and provide an additional filter at either the destination IP address (if defined) or a layer below the IP layer (e.g., Port and/or MPEG section filtering).

destination_ip_address The destination IP address associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the destination IP address of the packet and provide an additional filter at either the source IP address (if defined) or a layer below the IP layer (e.g., Port and/or MPEG section filtering).

source_port_number The UDP source port number associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the source UDP header and provide an additional filter at a layer below the UDP layer (e.g., MPEG section filtering).

destination_port_number The UDP destination port number associated with the download applicable to the Host. This is utilized to allow the Host to better filter packets in the tunnel. If the value is zero, then the Host SHALL ignore the UDP header and provide an additional filter at a layer below the UDP layer (e.g., MPEG section filtering).

application_id A DSG data stream identifier, associated with Type 4 Broadcast Tunnel information. The application_id value SHALL be greater than 0. Applicable only for the Host operating in DSG Advanced mode. This is utilized to allow the Host to build a DSG data stream routing table. This application ID SHALL be associated with the filter setting for a Type 4 Broadcast Tunnel, passed to the Host in the ***configure_advanced_DSG()*** APDU.

PID A DSG data stream identifier, associated with DSG Tunnel information. Applicable only for the Host operating in DSG Basic mode. This is utilized to

	allow the Host to open the MPEG section service_type data flow, from the Card to the Host, to receive an MPEG sections with code object delivered to the Card from DSG Broadcast Tunnel.
tftp_server_address	The IP address of the TFTP server where the code image resides. The code file name, as defined via the code_file_name_byte field, contains the complete directory path and name of the file to download. The address is 128 bits in length to support IPv6.
code_file_name_length	Length of code file name.
code_file_name_byte	Name of software upgrade file on carousel. This is the name of the Code File (see [SCTE23-2]) that is on the broadcast carousel as well as in Host Flash. For download_type = 0x00, 0x01, and 0x02, 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 ***code_version_table()*** APDU code_file_name_byte loop and the associated byte in the Download Info Indication message module Info Byte loop SHALL be the same.
number_of_cv_certificates	The number of code verification certificates.
certificate_type	Determines the type of CVC 0x00 Manufacturer CVC 0x01 Co-Signer CVC 0x02 - 0xFF Reserved
code_verification_certificate	Code Verification Certificate per [SCTE23-2].

E.6.4 code_version_table_reply()

The Type 1 Version 1, Type 1 Version 2 and Type 1 Version 3 of the ***code_download_table_reply()*** APDU is the same as defined in section 6.2.4 of [CDL].

E.6.5 host_download_control()

The Type 1 Version 1, Type 1 Version 2 and Type 1 Version 3 of the ***host_download_control()*** APDU is the same as defined in section 6.2.3 of [CDL].

E.6.6 host_download_command() Type 1 Version 1 (Deprecated)

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 E.6-5 - host_download_command (Type 1 Version 1)

Syntax	No. of bits	Mnemonic
host_download_command() {		
host_download_control_tag	24	uimsbf
length_field()		
host_command	8	uimsbf
location_type	8	uimsbf
if(location_type == 00){		
source_id	16	uimsbf
}		
if(location_type == 01){		
frequency_vector	16	uimsbf
transport_value	8	uimsbf
stream_ID	8	uimsbf

Syntax	No. of bits	Mnemonic
if(stream_ID == 00){		
Reserved	3	uimsbf
PID	13	uimsbf
}		
if(stream_ID == 01){		
program_number	16	uimsbf
}		
}		
}		

host_download_control_tag Value = 0x9F9C05

host_command Defines the priority of download.

- 0x00 Check for download during next cycle using source_id
- 0x01 Download now
- 0x02 Deferred download
- 0x03 Download now, no exceptions
- 0x04-0xFF Reserved

location_type Defines the method in which the Host device is to utilize to acquire the DSM-CC stream.

- 0x00 Transport Stream location is defined in the channel map and may be found via the defined source ID.
- 0x01 Indicates that the Host device is to use the frequency, modulation type, and PID or program number to acquire the DSM-CC stream.
- 0x02 - 0xFF Reserved

source_id 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.

frequency_vector 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.

transport_value Defined values as follows:

- 0x00 DOCSIS channel
- 0x01 FAT channel/QAM64
- 0x02 FAT channel/QAM256
- 0x03-0xFF Reserved

stream_ID Defines the way in which the DSM-CC is to be located.

- 0x00 Indicates that the DSM-CC stream is to be located utilizing the defined PID.
- 0x01 Indicates that the stream is to be located utilizing the program number.
- 0x02 - 0x FF Reserved

PID Packet identifier of the stream that contains the code file.

program_number Defines the program number in which the DSM-CC stream resides.

For Type 1 Version 2 and Type 1 Version 3, the ***host_download_command()*** APDU was removed, as CVDT signaling is not supported.

E.7 Extended Channel

This resource has six versions: Version 1 of this resource is required for Hosts that do not have an embedded High Speed Host (DOCSIS) Modem; Version 2 of this resource is required for Hosts that do have an embedded High Speed Host (DOCSIS) Modem and support the DSG basic mode; Version 3 of this resource is 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 is required for Hosts that have an embedded High Speed Host (DOCSIS) Modem and support both Basic and Advanced DOCSIS Set-top Gateway (DSG) with the additional message_type 0x05, eCM reboot, in the ***DSG_message()*** APDU. Version 5 modifies the APDUs to support IPv6. Version 6 is defined in Section 9.14 of this specification. Version 4 of this resource includes support for all of the objects defined by versions 3, 2, and 1. Version 6 of this resource includes support for all of the objects defined by version 5.

Table E.7-1 - Extended Channel Resource

Resource	Mode	Class	Type	Version	Identifier (hex)
Extended Channel	S-Mode/ M-Mode	160	1	1	0x00A00041
Extended Channel	S-Mode/ M-Mode	160	1	2	0x00A00042
Extended Channel	S-Mode/ M-Mode	160	1	3	0x00A00043
Extended Channel	S-Mode/ M-Mode	160	1	4	0x00A00044
Extended Channel	S-Mode/ M-Mode	160	1	5	0x00A00045

Unless otherwise indicated, the APDUs for a version are the same as in Section 9.14 of this specification.

E.7.1 new_flow_req() Type 1 Version 1 and Type 1 Version 2

Table E.7-2 - new_flow_req APDU (Type 1 Version 1 and Type 1 Version 2)

Syntax	No. of Bits	Mnemonic
new_flow_req() {		
new_flow_req_tag	24	uimsbf
length_field()		
service_type	8	uimsbf
if (service_type == mpeg_section) {		
Reserved	3	bslbf
PID	13	uimsbf
}		
if (service_type == ip_u) {		
MAC_address	48	uimsbf
option_field_length	8	uimsbf
for (i=0; i<option_field_length; i++) {		
option_byte	8	uimsbf
}		
}		
if (service_type == ip_m) {		
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 RFC 2132 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 28-bit 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 IP_multicast_group_ID address.

Table E.7-3 - new_flow_req APDU (Type 1 Version 3 and Type 1 Version 4)

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.

E.7.2 new_flow_cnf()

Table E.7-4 - new_flow_cnf APDU (Type 1 Version 1)

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 == 0) {		
flow_id	24	uimsbf
service_type	8	uimsbf
if (service_type == IP_U) {		
IP_address	32	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
- 0x05-0xFF 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.

Table E.7-5 - new_flow_cnf APDU (Type 1 Versions 2, 3 & 4)

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. To avoid conflicts between the assignment of flow_ids between the Card and the Host, the Card SHALL assign flow_ids in the range of 0x000001 to 0x7FFFFF, and the Host SHALL assign flow_ids in the range of 0x800000 to 0xFFFFFF. 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 E.7-6.

Bit 0 no_frag
bits 2:1 reserved

Table E.7-6 - Flag field definitions

BITS		
2	1	0
reserved		no_frag

no_frag A 1-bit Boolean that designates if the network supports fragmentation. A value of 0_2 indicates that fragmentation is supported. A value of 1_2 indicates that fragmentation is not supported.

max_pdu_size A 13-bit unsigned integer number that designates the maximum PDU length that may be transmitted across the interface.

option_field_length An 8-bit unsigned integer number that represents the number of bytes of option field data to follow.

option_byte These bytes correspond to the options requested in the ***new_flow_req()*** message. The format of the field is as defined in [RFC2132]. The end option (code 255) SHALL NOT be used.

Table E.7-7 - new_flow_cnf APDU Syntax (Type 1 Version 5)

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
}		
if (service_type == Socket) {		
reserved	3	uimsbf
max_pdu_size	13	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 Request denied, DNS not supported
0x07 Request denied, DNS lookup failed
0x08 Request denied, local port already in use or invalid
0x09 Request denied, could not establish TCP connection
0x0A Request denied, IPv6 not supported
0x0B-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. To avoid conflicts between the assignment of flow_ids between the Card and the Host, the Card and the Host will assign Ids in different ranges. The flow id value of 0x000000 is reserved and should not be assigned by either the Host or the Card.

The Card SHALL assign Extended Channel flow_ids in the range of 0x000001 to 0x7FFFFF in the ***new_flow_cnf()*** APDU.

The Host SHALL assign Extended Channel flow_ids in the range of 0x800000 to 0xFFFFFF in the ***new_flow_cnf()*** APDU.

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 This field is not supported in any version of the extended channel resource.

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 9.14-5.

Bit 0 no_frag
bits 2:1 reserved

Table E.7-8 - Flag field definitions

BITS		
2	1	0
reserved		no_frag

no_frag A 1-bit Boolean that designates if the network supports fragmentation. A value of 0_2 indicates that fragmentation is supported. A value of 1_2 indicated that fragmentation is not supported.

max_pdu_size A 13-bit unsigned integer number that designates the maximum PDU length that may be transmitted across the interface.

option_field_length An 8-bit unsigned integer number that represents the number of bytes of option field data to follow.

option_byte These bytes correspond to the options requested in the ***new_flow_req()*** message. The format of the field is as defined in [RFC2132].

The device replying with the ***new_flow_cnf()*** APDU SHALL NOT use the "end option" (code 255) in the option_byte field.

E.7.3 delete_flow_req()

The Type 1 Version 1, Type 1 Version 2, Type 1 Version 3, Type 1 Version 4, and Type 1 Version 5 of the ***delete_flow_req()*** APDU is the same as defined in Section 9.14.3.

E.7.4 delete_flow_cnf()

The Type 1 Version 1, Type 1 Version 2, Type 1 Version 3, Type 1 Version 4, and Type 1 Version 5 of the ***delete_flow_cnf()*** APDU is the same as defined in Section 9.14.4.

E.7.5 lost_flow_ind()

Table E.7-9 - lost_flow_ind APDU (Type 1 Versions 1, 2, 3, 4 and 5)

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 Remote TCP socket closed (V5 only)
- 0x05 Socket read error (V5 only)
- 0x06 Socket write error (V5 only)
- 0x04-0xFF Reserved (V1-4)
- 0x07-0xFF Reserved (V5)

E.7.6 lost_flow_cnf()

The Type 1 Version 1, Type 1 Version 2, Type 1 Version 3, Type 1 Version 4, and Type 1 Version 5 of the ***lost_flow_cnf()*** APDU is the same as defined in Section 9.14.6.

E.8 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:

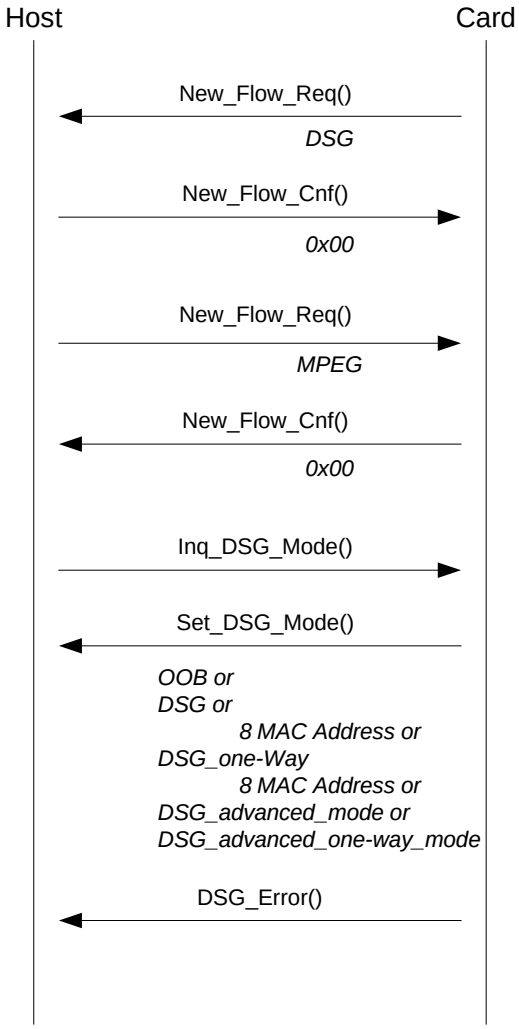


Figure E.8-1 - DSG Mode Message Flow

E.8.1 inquire_DSG_mode()

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 E.8-1 - inquire_DSG_mode APDU Syntax (Type 1 Versions 2, 3, and 4)

Syntax	No. of Bits	Mnemonic

inquire_DSG_mode() {
  inquire_DSG_mode_tag
  length_field()
}
	24	uimsbf

inquire_DSG_mode_tag 0x9F8E06

E.8.2 set_DSG_mode()

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 E.8-2 - set_DSG_mode APDU Syntax (Type 1 Versions 2, 3, and 4)

Syntax	No. of Bits	Mnemonic

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<number_MAC_addresses; i++) {
  DSG_MAC_address
  }
  remove_header_bytes
  }
}
	24
	
	8
	
	8
	
	48
	
	16
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf
		
		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

number_MAC_addresses	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.
DSG_MAC_address	The Ethernet MAC addresses allocated by the Card provider to carry the DSG tunnels.
remove_header_bytes	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 a received DCD message 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 or in the event of an eCM reset. 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:

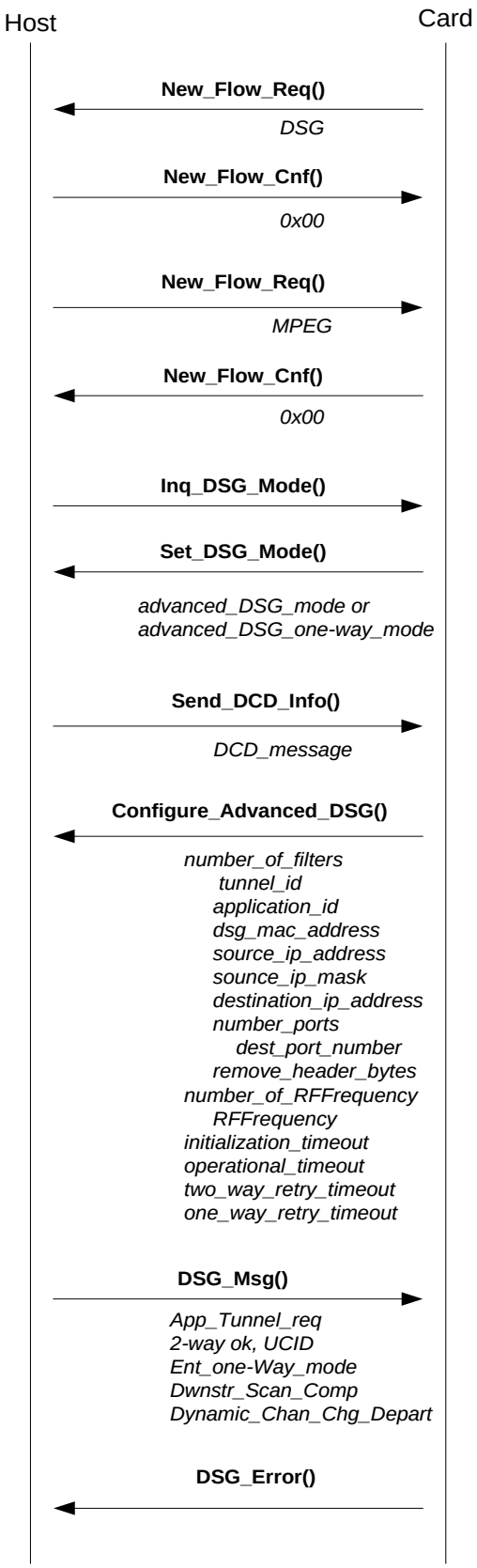


Figure E.8-2 - Sample Advanced Mode Message Flow

E.8.3 DSG_packet_error()

Table E.8-3 - DSG_packet_error (Type 1 Version 2)

Syntax	No. of Bits	Mnemonic

DSG_packet_error() {
  DSG_packet_error_tag
  length_field()
  error_status
}
	
	24
	
	8
		
		uimsbf
		
		uimsbf

lost_flow_ind_tag 0x9F8E08

error_status The Card can inform the Host of errors that occur in receiving DSG packets. The error_status indicates the type of error that occurred.

0x00 byte_count_error
0x01-0xFF Reserved

For Type 1 Version 3 and Type 1 Version 4 the ***DSG_packet_error()*** APDU message was renamed to ***DSG_error()*** APDU as defined below.

The Card MAY inform the Host of errors that occur in receiving DSG packets using the ***DSG_error()*** APDU.

Table E.8-4 - DSG_error APDU Syntax (Type 1 Version 3 and Type 1 Version 4)

Syntax	No. of Bits	Mnemonic

DSG_error() {
  DSG_error_tag
  length_field()
  error_status
}
	
	24
	
	8
		
		uimsbf
		
		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

E.8.4 configure_advanced_DSG()

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 E.8-5 - Configure Advanced DSG Object Syntax (Type 1 Version 3 and Type 1 Version 4)

Syntax	No. of Bits	Mnemonic
configure_advanced_DSG () {		
configure_advanced_DSG _tag	24	uimsbf
length_field()		
number_of_filters	8	uimsbf
for (i=0; i< number_tunnel_filters; i++) {		
tunnel_id	8	uimsbf
application_id	16	uimsbf
dsg_mac_address	48	uimsbf
source_IP_address	32	uimsbf
source_IP_mask	32	uimsbf
destination_IP_address	32	uimsbf
number_ports	8	uimsbf
for(i=0; i< number_ports; i++){		
dest_port_number	16	uimsbf
}		
remove_header_bytes	16	uimsbf
}		
number_of_RXFrequency	8	uimsbf
for (i=0; i<number_of_RXFrequency; i++){		
RXFrequency	32	uimsbf
}		
initialization_timeout	16	uimsbf
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.

E.8.5 DSG_Message()

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, to indicate that the eCM has received a DCC-REQ message and is preparing to execute a Dynamic Channel Change, or to indicate that an event has occurred that required an eCM reboot.

Table E.8-6 - DSG Message Object Syntax (Type 1 Version 3)

Syntax	No. of bits	Mnemonic
dsg_message () {		
dsg_message_tag	24	uimsbf
length_field()		
message_type	8	uimsbf
If (message_type = 0x00) {		
number_app_ids	8	uimsbf
for (i=0; i < number_app_ids; i++) {		
application_id	16	uimsbf
}		
}		
If (message_type = 0x01) {		
UCID	8	uimsbf
}		
if (message_type = 0x04) {		
init_type	8	uimsbf
}		
}		

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. The Card uses this value for filtering of various DSG rules as applicable.
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 as an indicator that the eCM has been unable to identify a downstream channel with a DCD message after a complete downstream scan.

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 channels(s), as directed by the CMTS.
0 = Reinitialize the MAC
1 = Perform broadcast initial ranging on new channel before normal operation
2 = Perform unicast initial ranging on new channel before normal operation
3 = Perform either broadcast initial ranging or unicast initial ranging on new channel before normal operation
4 = Use the new channel(s) directly without re-initializing or initial ranging
5 = Reinitialization method not specified
6 - 255: reserved

0x05 - 0xFF Reserved

Table E.8-7 - DSG Message Object Syntax (Type 1 Version 4)

Syntax	No. of bits	Mnemonic

dsg_message () {
dsg_message_tag
length_field()
message_type
If (message_type = 0x00) {
number_app_ids
for (i=0; i < number_app_ids; i++) {
application_id
}
}
If (message_type = 0x01) {
UCID
}
if (message_type = 0x04) {
init_type
}
}
	
	24
	
	8
	
	8
	
	16
	
	8
	
	8
	
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf

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 channels(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

0x05 eCM Reset - an event has occurred that required an eCM reboot. The Card needs to re-establish DSG tunnel filtering by sending the ***configure_advanced_DSG()*** object. The tunnel MAC address and DSG classifiers can be obtained by parsing the next received DCD message or from a local cache.

0x06 - 0xFF Reserved

E.8.5.1 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.

E.8.6 send_DCD_info()

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 E.8-8 - send_DCD_info Object Syntax (Type 1 Version 3 and Type 1 Version 4)

Syntax	No. of bits	Mnemonic

send_DCD_info () {
send_DCD_info _tag
length_field()
DCD_message
}
	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.

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

The following ECNs were incorporated into OC-SP-CCIF2.0-I03-051117:

ECN	Description	Date
CCIF2.0-N-05.0761-6	Modifications to Common Download to Support Delivery via DSG Broadcast Tunnel	7/15/05
CCIF2.0-N-05.0800-3	Conflict resolution between Host flow ids and Card flow ids	9/9/05
CCIF2.0-N-05.0807-1	LogicCB requirement	9/14/05
CCIF2.0-N-05.0808-1	Resource Manager Legacy Support	9/23/05
CCIF2.0-N-05.0809-1	FDC_Status_Report Correction	10/21/05
CCIF2.0-N-05.0810-1	Update to RF_TX_Rate_Value	9/6/05
CCIF2.0-N-05.0811-1	Delete of DLS System Time APDU	9/6/05
CCIF2.0-N-05.0813-4	Changes to OOB Interface description of DSG to accurately define header byte removal	9/23/05
CCIF2.0-N-05.0814-2	DII Message Corrections	10/7/05
CCIF2.0-N-05.0815-2	Deletion of open_MMI_cnf() APDU for M-Mode	10/7/05
CCIF2.0-N-05.0816-2	Reference and editorial updates	10/17/05
CCIF2.0-N-05.0823-1	M-Mode Device Capability Discovery Clarifications	10/31/05

The following ECNs were incorporated into OC-SP-CCIF2.0-I04-060126:

ECN	Description	Date
CCIF2.0-N-05.0821-6	Extend Generic Diagnostic Resource Capability	1/13/06
CCIF2.0-N-05.0831-1	n-Band_tune_req() APDU update	12/2/05
CCIF2.0-N-05.0833-1	Remove Low Speed Session Open Requirement	12/2/05
CCIF2.0-N-05.0838-1	Card READY/SDO behavior when invalid VPP1/VPP2 is detected	12/29/05
CCIF2.0-N-05.0850-1	Interface Query Byte Data Exchange Clarification	1/13/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I05-060413:

ECN	Description	Date
CCIF2.0-N-05.0849-4	Additions/Corrections to the CVT	3/3/06
CCIF2.0-N-05.0854-1	Clarify DSG operation after eCM reboot	1/27/06
CCIF2.0-N-05.0857-1	Card Signal Timing Parameter Connection	1/27/06
CCIF2.0-N-06.0875-1	Poll Time-out Timer Correction	3/17/06
CCIF2.0-N-06.0876-1	Error Code Update	3/17/06
CCIF2.0-N-06.0878-1	Revised description of the download_type	3/24/06
CCIF2.0-N-06.0879-1	COR Write Timing Correction	3/30/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I06-060622:

ECN	Description	Date
CCIF2.0-N-06.0882-1	Modify M-Card Maximum Power	5/8/06
CCIF2.0-N-06.0886-4	Open Session Request Correction	6/9/06
CCIF2.0-N-06.0888-1	Clarification of System Time	5/18/06
CCIF2.0-N-06.0889-4	CVT update for self-identification of its resource version	6/9/06
CCIF2.0-N-06.0899-2	M-Mode Error Code Updates	6/9/06
CCIF2.0-N-06.0900-1	Clarification of Private Resource Identifier	6/9/06
CCIF2.0-N-06.0905-1	Copy Protection Resource Version Change	6/13/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I07-060803:

ECN	Description	Date
CCIF2.0-N-06.0895-5	Modifications to Card/Host IP Model	7/21/06
CCIF2.0-N-06.0883-10	New Advanced DSG Resource Type	8/2/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I08-061031:

ECN	Description	Date
CCIF2.0-N-06.0909-4	Generic Diagnostics Corrections	9/8/06
CCIF2.0-N-06.0914-1	Application_info_cnf() and Server_query() APDU URL Clarification	8/11/06
CCIF2.0-N-06.0915-2	Removal of CDL from CCIF 2.0	10/13/06
CCIF2.0-N-06.0916-3	Legacy Resource/ APDU Annex Addition	10/13/06
CCIF2.0-N-06.0924-4	Interface Resource Loading Clarification	10/13/06
CCIF2.0-N-06.0925-1	Clarification of DSG Count	9/8/06
CCIF2.0-N-06.0932-2	Modify text of Annex B error code 161-64 for M-Card	10/13/06
CCIF2.0-N-06.0933-2	Add requirements for versions 2, 3, and 4 of the extended channel	10/13/06
CCIF2.0-N-06.0935-2	SAS Editorial Update	10/13/06
CCIF2.0-N-06.0937-2	UDP and TCP protocols support for IP Unicast Service Flow	10/13/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I09-070105:

ECN	Description	Date
CCIF2.0-N-06.0941-4	Generic_Feature_Control feature enhancements	12/8/06
CCIF2.0-N-06.0942-1	Move Copy Protection Resource identifier table to CCCP2.0	11/10/06
CCIF2.0-N-06.0948-1	Removal of CDL Appendices I and II	12/22/06
CCIF2.0-N-06.0949-1	Clarification on Application of CCI	12/22/06
CCIF2.0-N-06.0950-1	FR bit Timing Clarification	12/22/06
CCIF2.0-N-06.0957-2	Do not require one-way operation after forwarding is restricted	12/22/06
CCIF2.0-N-06.0961-1	Fix to CableCARD's DHCP Option 43	12/22/06

The following ECNs were incorporated into OC-SP-CCIF2.0-I10-070323:

ECN	Description	Date
CCIF2.0-N-06.0964-2	Editorial corrections on the DSG Resource	2/5/07
CCIF2.0-N-06.0966-1	Remove Generic Features Storage Requirement	2/5/07
CCIF2.0-N-06.0967-1	Time_zone_offset related comment deletion	2/23/07
CCIF2.0-N-06.0974-3	Socket Flows requirement clarification	3/9/07
CCIF2.0-N-07.0978-1	Clarifications of ca_pmt APDUs for IPPV program support	2/23/07
CCIF2.0-N-07.0990-2	Adding vct_id parameter to generic features	3/9/07
CCIF2.0-N-07.0998-2	Host Reset Vector	3/9/07
CCIF2.0-N-07.0999-1	Adding turn-on channel parameter to generic features	3/9/07
CCIF2.0-N-07.1002-3	Add requirement to disallow filtering by Host for Card requested transport streams	3/9/07

The following ECNs were incorporated into OC-SP-CCIF2.0-I11-070615:

ECN	Description	Date
CCIF2.0-N-07.0989-3	Clarification of Command and Extended Channel S-Mode Operation	6/1/07
CCIF2.0-N-07.1021-2	Clarification of DSG Directory	6/1/07
CCIF2.0-N-07.1023-1	Extended Channel resource V5 DSG resource implementation	6/1/07
CCIF2.0-N-07.1030-3	Remove mandatory support for Headend Communication resource in Card M-Mode	4/23/07
CCIF2.0-N-07.1044-3	Resource version support and reporting	6/1/07

The following ECNs were incorporated into OC-SP-CCIF2.0-I12-071113:

ECN	Description	Date
CCIF2.0-N-07.1022-4	Improvement In Interface Error Recovery	10/5/07
CCIF2.0-N-07.1043-4	Generic Feature Control Features Clarifications	8/31/07
CCIF2.0-N-07.1051-2	Extended Channel Resource Max Session Correction	8/10/07
CCIF2.0-N-07.1065-1	Omit 'Card reset' in firmware_upgrade_complete() APDU for M-Mode	8/10/07
CCIF2.0-N-07.1068-2	Modify PCMCIA Card length requirement	10/5/07
CCIF2.0-N-07.1073-4	ReqPro Edits to Section 9	8/31/07
CCIF2.0-N-07.1080-1	The direction of inquire_DSG_mode() and set_DSG_mode() APDUs needs to be corrected	8/31/07
CCIF2.0-N-07.1081-1	Extended Channel Flow Requirements table needs to be corrected	8/31/07
CCIF2.0-N-07.1082-5	Host Addressable Properties	10/23/07
CCIF2.0-N-07.1088-5	Adding terminal association parameter to generic features	10/23/07
CCIF2.0-N-07.1091-1	M-Card CPU interface DA flag ambiguity	10/23/07
CCIF2.0-N-07.1093-3	Group Download Generic Features Support	10/23/07
CCIF2.0-N-07.1095-2	DSG Resource - Host processing of multiple instances of the same broadcast type	10/23/07
CCIF2.0-N-07.1101-3	Adding ZIP code to generic features	10/23/07
CCIF2.0-N-07.1104-2	Adding Length Values to Generic Feature Control	10/23/07
CCIF2.0-N-07.1117-1	DSG_directory Editorial Correction	10/23/07

The following ECNs were incorporated into OC-SP-CCIF2.0-I13-080118:

ECN	Description	Date
CCIF2.0-N-07.1120-2	IP Address Support and Clarification of Lost Flows	12/21/07
CCIF2.0-N-07.1132-2	IP Unicast Flow Requirements Clarification	12/21/07
CCIF2.0-N-07.1143-2	Omnibus part 2 edits for ReqPro format	12/21/07
CCIF2.0-N-07.1146-1	Generic Feature daylight savings clarification	12/21/07

The following ECNs were incorporated into OC-SP-CCIF2.0-I14-080404:

ECN	Description	Date
CCIF2.0-N-07.1158-1	Clarification on Generic Feature Control feature_list() APDU	2/29/08
CCIF2.0-N-07.1160-1	Default OOB Mode Defined	2/29/08
CCIF2.0-N-08.1168-1	Addition of Error Handling Item	2/29/08
CCIF2.0-N-08.1169-2	Correct Figure 5.10-3	2/29/08
CCIF2.0-N-08.1192-1	New flow req table syntax	3/14/08

The following ECNs were incorporated into OC-SP-CCIF2.0-I15-080620:

ECN	Description	Date
CCIF2.0-N-08.1195-2	Host Reset Vector	4/18/08
CCIF2.0-N-08.1223-1	Adding CANH to Application Information	5/30/08
CCIF2.0-N-08.1224-1	Clarification of CableCARD thermal environment	5/30/08
CCIF2.0-N-08.1245-2	Generic Features Parameters Update and Support	5/30/08

The following ECNs were incorporated into OC-SP-CCIF2.0-I16-081114:

ECN	Description	Date
CCIF2.0-N-08.1267-4	Retrieval of CableCARD MIB Objects	10/17/08
CCIF2.0-N-08.1293-2	Deprecate DSG Basic Mode	10/17/08
CCIF2.0-N-08.1312-1	Clarify use of socket flow fields new_flow_conf() APDU	10/17/08
CCIF2.0-N-08.1313-3	DSG support in S-Mode	10/17/08

The following ECNs were incorporated into OC-SP-CCIF2.0-I17-090206:

ECN	Description	Date
CCIF2.0-N-08.1358-1	Clarification on source_IP_mask	1/16/09
CCIF2.0-N-08.1363-2	Correct MfgId to Sync with Host MIB	1/16/09

The following ECNs were incorporated into OC-SP-CCIF2.0-I18-090508:

ECN	Description	Date
CCIF2.0-N-08.1212-2	Resource and APDU Tag Reservation	4/17/09
CCIF2.0-N-09.1377-1	System Time Clarification	4/17/09
CCIF2.0-N-09.1379-3	Application_info_cnf() APDU Type 2 Version 1 and Version 2 Table Syntax	4/17/09

The following ECNs were incorporated into OC-SP-CCIF2.0-I19-090904:

ECN	Description	Date
CCIF2.0-N-08.1238-8	Clarify the use of CA_PMT upon selection of new programs	6/2/09
CCIF2.0-N-09.1402-2	Clarifications to UDP packet filtering in Socket Flow	8/7/09
CCIF2.0-N-09.1406-1	Update to Resource Manager to reduce number of sessions to one	8/7/09
CCIF2.0-N-09.1408-2	Addition of Vendor ID to Generic Diagnostics	8/7/09
CCIF2.0-N-09.1411-3	Clarification of Extended Channel IP_U Operation	8/7/09
CCIF2.0-N-09.1415-2	Clarify SAS Session maintenance	8/7/09

The following ECNs were incorporated into OC-SP-CCIF2.0-I20-091211:

ECN	Description	Date
CCIF2.0-N-09.1442-1	snmp_req() Requirement Update for Card MIB Access Support	11/6/09
CCIF2.0-N-09.1446-1	Fix Card DHCP option 43	11/6/09
CCIF2.0-N-09.1447-2	CCIF Omnibus ECNs corrections	11/20/09

The following ECNs were incorporated into OC-SP-CCIF2.0-I21-100507:

ECN	Description	Date
CCIF2.0-N-10.1500-3	Handling Multiple CA Descriptors in S-Mode and M-Mode	4/16/10
CCIF2.0-N-10.1511-3	Clarify SNMP SNMP_REQ() APDU content	4/30/10

The following ECN was incorporated into OC-SP-CCIF2.0-I22-100910:

ECN	Description	Date
CCIF2.0-N-10.1549-3	Clarification of Generic Feature DST Operation	7/16/10

The following ECN was incorporated into OC-SP-CCIF2.0-I23-110512:

ECN	Description	Date
CCIF2.0-N-11.1642-2	SAS CableCARD Reset	5/9/12
CCIF2.0-N-11.1644-1	Send only new DSG filter configurations to eCM	5/9/12
CCIF2.0-N-11.1656-2	CCIF Reference Edits	5/9/12

The following ECNs were incorporated into OC-SP-CCIF2.0-I24-120112:

ECN	Description	Date
CCIF2.0-N-11.1673-1	CCIF changes for DSG Set-top Extender Bridge operation	8/12/11
CCIF2.0-N-11.1688-2	Remove conflicting requirement on the M-Mode MPEG Transport Stream pre-header	9/23/11
CCIF2.0-N-11.1697-1	Deprecation of Socket Flow remote_addr_type option name DNS	11/4/11
CCIF2.0-N-11.1734-1	Deprecate ADSG IP_U flows	12/16/11

The following ECN was incorporated into OC-SP-CCIF2.0-I25-120531:

ECN	Description	Date
CCIF2.0-N-12.1779-1	Correct MfgId to include current CableCARDs	5/18/12

The following ECN was incorporated into OC-SP-CCIF2.0-I26-130418:

ECN	Author	Date	Description
CCIF2.0-N-13.1823-1	Skinner	4/5/13	Clarification of send_DCD_info behavior to account for Downstream Frequency Override