OpenCable Specifications

Tuning Resolver Interface Specification

OC-SP-TRIF-I03-090206

ISSUED

Notice

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

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

Document Status Sheet

Document Control Number:	OC-SP-TRIF-I03-090206			
Document Title:	Tuning Resolver Interface Specification			
Revision History:	I01 - Released 1/30/08 I02 - Released 5/19/08 I03 - Released 2/6/09			
Date:	February 6, 2009			
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.

Trademarks:

Contents

Figures

Tables

This page intentionally left blank.

1 SCOPE AND PURPOSE

1.1 Scope

This document provides technical details on the communications interface between Unidirectional Digital Cable Products (UDCPs) and a Tuning Resolver (TR).

1.2 Purpose

The FCC requirements for UDCPs prohibit the inclusion of reverse transmitters. Modern cable systems are now providing more and more programming through the use of a technology known as switched digital video (SDV). This enhanced service requires the use of direct communications through the cable plant to the headend using reverse transmitters and proprietary signaling of the cable systems. UDCPs built according to the applicable FCC rules are unable to access such services due to the prohibition on reverse transmitters within the product.

A Tuning Resolver was developed as a way to enable access to SDV services on UDCP products. This TR provides the necessary reverse transmitter and private signaling necessary to communicate SDV tuning requests to the headend. It "resolves" the tuning information for every tune-request of UDCP and provides detailed information to the UDCP to enable access to these services.

The TR connects in-line with the RF signal from the cable system to the UDCP and utilizes a USB interface for additional communications. This specification provides details on that communications interface and the signaling and messages used.

This document does not describe any details of the signaling used between the TR and the cable headend, which may vary from one system to another.

2 REFERENCES

2.1 Normative References

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 could be required to use or implement such normative references.

[CCCP] CableCARD™ Copy Protection 2.0 Specification, OC-SP-CCCP2.0-I08-071113, November 13, 2007, Cable Television Laboratories, Inc.

[CCIF2.0] CableCARD Interface 2.0 Specification, OC-SP-CCIF2.0-I17-090206, February 6, 2009, Cable Television Laboratories, Inc.

[MINI-B] USB 2.0 Specification Engineering Change Notice (ECN) #1: Mini-B connector, 10/20/2000.

[RFC 2104] IETF RFC 2104, HMAC: Keyed-Hashing for Message Authentication, February 1997.http://ietf.org/rfc/rfc2104.txt

[SCTE 18] SCTE 18 2007, Emergency Alert Messaging for Cable.

[SCTE 28] ANSI/SCTE 28 2007, HOST-POD Interface Standard.

[SCTE 65] ANSI/SCTE 65 2002, Service Information Delivered Out-of-Band for Digital Cable Television.

[USB2.0] Universal Serial Bus Specification Revision 2.0, April 27, 2000.

2.2 Informative References

This document uses the following informative references.

[SCTE 55-1] ANSI/SCTE 55-1 2002, Digital Broadband Delivery System: Out Of Band Transport Part 1: Mode A, http://www.scte.org/content/index.cfm?pID=59

[SCTE 55-2] ANSI/SCTE 55-2 2002, Digital Broadband Delivery System:Out Of Band Transport Part 2: Mode B, http://www.scte.org/content/index.cfm?pID=59

2.3 Reference Acquisition

- American National Standards Institute (ANSI), 1819 L Street, NW, Suite 600, Washington, DC 20036, Telephone +1 (202) 293-8020, http://webstore.ansi.org/ansidocstore/default.asp/
- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone 303-661-9100; Fax 303-661-9199; Internet: http://www.cablelabs.com /
- Internet Engineering Task Force (IETF), Internet: http://www.ietf.org
- SCTE, SCTE Headquarters, 140 Philips Rd., Exton, Pa 19341-1318, Phone: 610-363-6888, www.scte.org
- USB Implementers Forum, http://www.usb.org/developers/docs/

3 TERMS AND DEFINITIONS

This specification uses the following terms:

CableCARD™	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.
POD Module	Point of Deployment Module (a.k.a. CableCARD)
Resolver	Short for Tuning Resolver

4 ABBREVIATIONS, ACRONYMS AND CONVENTIONS

4.1 Abbreviations and Acronyms

This specification uses the following abbreviations:

APDU	Application Packet Data Unit
bslbf	Bit String, Leftmost Bit First
EAS	Emergency Alert System
HMAC	Hash Message Authentication Code
LTSID	Local Transport Stream ID
MMI	Man-Machine Interface
POD	Point Of Deployment
RF	Radio Frequency
SDV	Switched Digital Video
simsbf	Signed Integer, Most Significant Bit First
UDCP	Unidirectional Digital Cable Product
uimsbf	Unsigned Integer, Most Significant Bit First
USB	Universal Serial Bus
TR	Tuning Resolver

4.2 Conventions

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 or the adjective "OPTIONAL" 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.

5 OVERVIEW (INFORMATIVE)

The Tuning Resolver (TR) consists of the hardware and software required to allow a UDCP device to tune both linear and SDV programs using the interface defined in this specification. The UDCP will connect to the TR via a standard USB connector, with the UDCP as the USB Host device and the TR as the USB client device. Figure 1 shows this interface.

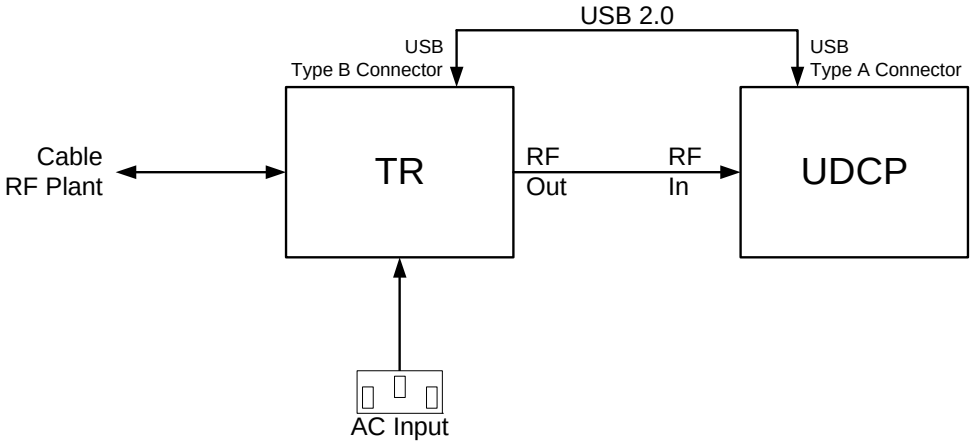

Figure 1 - UDCP/TR Interface

The TR contains whatever hardware is required for it to communicate with the headend (e.g., QPSK SCTE 55-1 or SCTE 55-2, DOCSIS, etc.) and to interface to the UDCP (i.e., USB). The block diagram in Figure 2 is an example of a proposed TR.

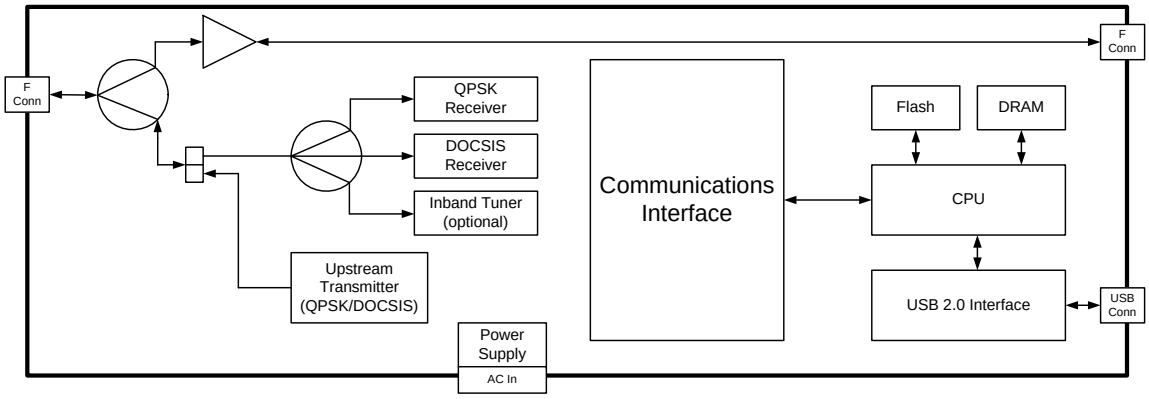

Figure 2 - TR Block Diagram Example

Note: If the TR manufacturer wants to support technology such as MOCA, the TR would be designed to have an RF insertion loss of less than 4 dB in each direction up to 1 GHz, and less than 6 dB up to 1.55 GHz.

The TR and CableCARD operate independently of each other.

5.1 RF Interface

The RF interface(s) of the TR are out of scope for this document.

5.2 USB Interface

The USB interface between the UDCP and TR will be a USB 2.0 compatible interface using the Data class.

5.3 Channel Tuning Operation

After the USB initialization and enumeration, the UDCP initiates the initialization and discovery process, during which the UDCP and TR discover each other's capabilities. The TR then sends the channel table to the UDCP in place of the [SCTE 65] SI data. The UDCP then utilizes this channel table to know which channels are defined. Whenever the UDCP tunes any channel, it sends the channel number to the TR, upon which the TR will, if necessary, handle all SDV interface operations for SDV channels, and return the tuning parameters required for the UDCP to tune the channel.

5.4 EAS Operation

In the event that the UDCP receives an EAS message (see [SCTE 18]), the UDCP could be required to tune based on the source ID instead of channel number. This is supported in the resolve_tuning_req() APDU.

5.5 Firmware Upgrade

5.5.1 CableCARD Firmware Upgrade Tuning

The CableCARD Host Control inband_tuning() APDU will either load the exact frequency and modulation value into the Host or will give the Host a source ID to tune. If it is the latter, then it is expected that the Host will use the TRIF resolve_tuning_req() APDU defined in 8.3.1 with the channel_source_type set to source ID instead of channel number.

5.5.2 TR Firmware Upgrade

The TR will inform the UDCP that it is in the process of doing a firmware upgrade by the tr_operational_status parameter, which is in the tr_status functions. A TR firmware upgrade may require that the TR reset. It is expected that the TR will do this using the standard USB client reset operation.

6 PHYSICAL INTERFACE (NORMATIVE)

The TR interface uses standard USB cabling to the UDCP and is defined below. The RF and power connections of the TR interface are outside the scope of this specification.

The following is a list of the physical requirements of the TR interface.

The UDCP SHALL be the USB Host.

The UDCP USB interface SHALL be the USB type A receptacle [USB2.0].

The TR SHALL be the USB client.

The TR USB interface SHALL be the USB type B receptacle [USB2.0] or an adapter that logically proxies a USB type B client interface.

The TR USB interface SHALL NOT use the USB mini-B receptacle [MINI-B].

The TR USB interface MAY draw power from the USB +5V signal.

The UDCP USB interface SHALL meet the full USB power requirements as defined in [USB2.0].

The TR USB interface SHALL NOT be designed such as to be dependent on utilizing the USB connector as a heat sink.

If the USB cable is not user-replaceable, the TR USB interface type A plug Overmold Boot SHALL NOT exceed 16 mm in width and 8 mm in height, as defined in Figure 6-9 of [USB2.0].

7 USB INTERFACE (NORMATIVE)

This section covers the minimum USB interface requirements.

The UDCP SHALL support either the USB 1.1 interface as defined in [USB2.0] or the USB 2.0 interface as defined in [USB2.0].

The TR Interface SHALL use the USB 2.0 interface as defined in [USB2.0]. This includes support for USB 1.1.

The TR Interface SHALL use Bulk type data transfers for transfer of the TR APDUs.

The TR SHALL implement full-speed bulk transfers. The TR MAY implement high-speed bulk transfers.

The UDCP SHALL implement full-speed bulk transfers. The UDCP MAY implement high-speed bulk transfers.

The TR SHALL send a single APDU in a single bulk transfer according to section 5.8 of [USB2.0].

The UDCP SHALL send a single APDU in a single bulk transfer according to section 5.8 of [USB2.0].

The TR SHALL be able to service bulk transfers at an average rate of not less than 100 kb/s.

The UDCP SHALL be able to service bulk transfers at an average rate of not less than 100 kb/s.

Note: There is a (remote) possibility of asymmetric transient behavior on the USB link. It is desirable for both physical endpoints to implement a bulk transfer error timeout to provide an error trap in the eventuality that a bulk transfer does not conclude in an expected period of time. For example, a 100 kb/s transfer rate should be expected to move a 65 kB channel table message from the TR to the UDCP in around six seconds. A 10-second transfer timeout seems reasonable for this large APDU packet example.

The TR SHALL use the value for 'USB 2.0' for the bcdUSB USB Device Descriptor.

The TR SHALL report one of the following pairs of values for idVendor number and idProduct number in the USB Device Descriptor: (idVendor = 0x05A6, idProduct = 0x0008) or (idVendor = 0x07B2, idProduct = 0x6002).

The TR SHALL report only one of the following values in the bDeviceClass field of the USB Device Descriptor: 0x00 (unspecified), 0x02 (Communications Device), or 0xFF (Vendor Specific).

The TR SHALL report only one of the following values in the bDeviceSubClass field of the USB Device Descriptor: 0x00 (unspecified), or 0xFF (Vendor Specific).

The TR SHALL report only one of the following values in the bDeviceProtocol field of the USB Device Descriptor: 0x00 (unspecified), or 0xFF (Vendor Specific).

The TR SHALL report bNumConfigurations in the USB Device Descriptor as greater than or equal to 1.

The UDCP SHALL select the first device configuration for communicating with the TR (i.e., bConfigurationValue 0x01 in USB Standard Configuration Descriptor).

The TR SHALL report bInterfaceClass in the USB Interface Descriptor as 0x0A (Data Interface Class) (only if the bDeviceClass is reported as 0x02), or 0xFF (Vendor Specific).

The TR SHALL report bNumEndpoints in the USB Interface Descriptor as 2.

The TR USB interface SHALL present two bulk transfer endpoints, one IN and one OUT in addition to the required default control endpoint 0 for standard USB requests. No descriptor is required for endpoint 0.

The UDCP SHALL support TRs that report the idVendor Numbers and idProduct numbers as listed in this spec.

The UDCP SHALL respond to a USB Reset with a complete re-initialization, including re-authentication.

If the TR is reset for any reason, it SHALL initiate a USB reset.

If the TR is not present on the USB interface at initialization, then the UDCP SHALL use [SCTE 65] for all tuning information.

If the TR USB connection is unavailable for any reason, then the UDCP SHALL discard the channel_table and start using the tuning information carried in [SCTE 65].

8 TR MESSAGE PROTOCOL (NORMATIVE)

This section defines the interface messages between the UDCP and TR.

The TR SHALL be able to interface and operate with a UDCP that is operating at a previous version of this specification.

The UDCP SHALL be able to interface and operate with a TR that is operating at a previous version of this specification.

Note: All previous versions will be documented in this specification.

8.1 Message Structure

A relatively simple message structure is utilized. It is based on the APDU format utilized in [CCIF2.0], with the exception that there is no session layer, the length field is different, and the APDU tag is 16 bits instead of 24 bits.

Table 1 defines the basic message structure:

Table 1 - Basic Format of TR Message

Syntax	# of bits	Mnemonic

trif_basic_message(){
  tr_message_tag
  length_field
  body_of_tr_message()
}
	
	16
	16
		
		uimsbf
		uimsbf

8.2 Initialization and Discovery Messages

The following table defines the Initialization and Discovery messages. These are only to be transmitted after USB enumeration is completed.

Table 2 - Initialization and Discovery Messages (Informative)

Message Name	Message Tag (hex)	Direction	Description
tr_init_req	01 01	UDCP → TR	UDCP requests TR initialization.
tr_init_rsp	01 02	UDCP ← TR	TR response to initialization request
challenge_req	01 05	UDCP ← TR	TR challenge to UDCP
challenge_rsp	01 06	UDCP → TR	UDCP response to TR challenge
channel_table_req	01 07	UDCP → TR	UDCP requests channel table
channel_table_rsp	01 08	UDCP ← TR	TR sends channel table
channel_table_update	01 09	UDCP ← TR	TR sends channel table update
tr_hmac_key_send	01 0A	UDCP ← TR	TR HMAC Key to UDCP

The TR MAY send the channel_table_update unsolicited at any time after initialization.

The UDCP SHALL switch to using a new channel table within five minutes of receiving a channel_table_update message.

8.2.1 tr_init_req()

The UDCP SHALL send the tr_init_req() APDU as defined in Table 3 after the USB enumeration has completed.

Table 3 - TR Initialization Request APDU

Syntax	# of bits	Mnemonic
tr_init_req(){		
tr_init_req_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
udcp_profile()		
}		

tr_init_req_tag 0x0101

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

udcp_profile() As defined in section 8.6.2.

8.2.2 tr_init_rsp()

The TR SHALL send the tr_init_rsp() APDU as defined in Table 4 within five seconds after receiving the tr_init_req() APDU.

If the UDCP does not receive the tr_init_rsp() APDU within five seconds after sending the tr_init_req() APDU, then the UDCP SHALL perform a USB reset and try sending the tr_init_req() APDU again.

After two failures of receiving the tr_init_rsp() APDU, the UDCP MAY declare the TR inoperable and display a message stating such to the user.

Table 4 - TR Initialization Response APDU

Syntax	# of bits	Mnemonic
tr_init_rsp(){		
tr_init_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
revision_status	8	uimsbf
tr_profile()		
}		

tr_init_rsp_tag 0x0102

trif_revision_code 0x01

request_id The value in the tr_init_req() APDU.

revision_status The status of the received revision.

0x00 – Revision in tr_init_req() APDU supported

0x01-0xFF – Highest revision supported by TR

tr_profile() As defined in section 8.6.3.

8.2.3 challenge_req()

The TR SHALL send a challenge_req() APDU as defined in Table 5 to authenticate the UDCP.

The TR SHALL send the challenge_req() APDU within five seconds of sending the tr_init_rsp() APDU.

The TR MAY send an unsolicited challenge_req () APDU to the UDCP at any time.

The TR SHALL restrict the datatype_id values in the challenge_req() APDU to 7, 13, 15, and 17.

The TR SHALL include every datatype_id that it needs for authentication in a single challenge_req() APDU.

Table 5 - Challenge Request APDU

Syntax	# of bits	Mnemonic
challenge _req(){		
challenge_req_tag	16	Uimsbf
length_field	16	Uimsbf
trif_revision_code	8	Uimsbf
request_id	16	Uimsbf
request_datatype_nbr	8	Uimsbf
for (i=0; i<request_datatype_nbr; i++){		
datatype_id	8	Uimsbf
}		
}		

challenge_req_tag 0x0105

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

request_datatype_nbr As defined in section 11 of [CCCP].

datatype_id As defined in section 11 of [CCCP].

8.2.4 challenge_rsp()

The UDCP SHALL send a challenge_rsp() APDU as defined in Table 6 within five seconds of receiving a challenge_req() APDU.

Table 6 - TR Challenge Response APDU

Syntax	# of bits	Mnemonic
challenge_rsp(){		
challenge_rsp_tag	16	Uimsbf
length_field	16	Uimsbf
trif_revision_code	8	Uimsbf
request_id	16	Uimsbf
send_datatype_nbr	8	Uimsbf
for_(i=0; i<send_datatype_nbr; i++){		
datatype_id	8	Uimsbf
datatype_length	16	Uimsbf
for (j=0; j<datatype_length; j++){		
data_type	variable	
}		
}		
}		

challenge_rsp_tag 0x0106

trif_revision_code 0x01

request_id The value in the channel_req() APDU.

send_datatype_nbr As defined in section 11 of [CCCP].

datatype_id As defined in section 11 of [CCCP].

datatype_length As defined in section 11 of [CCCP].

data_type As defined in section 11 of [CCCP].

For datatype_id 13 the UDCP sends an existing DH_pubKey_H for a currently bound CableCARD. For a UDCP that is not bound to a CableCARD, the UDCP may compute a new DH_pubKey_H as described in Step 1 of Figure 5.3-1 of [CCCP], or the UDCP may create a 1024 bit random number according to the requirements in section 7.2 of [CCCP] on random number generation and use this 1024 bit random number unique to the TR interface in place of DH_pubKey_H.

The UDCP SHALL provide values in the challenge_rsp() APDU for all datatype ids that are specified in the associated challenge_req() APDU.

The UDCP SHALL restrict the datatype values in the challenge_rsp() APDU to 7, 13, 15 and 17.

The UDCP SHALL send every datatype requested in the challenge_req() APDU from the TR in a single challenge_rsp() APDU.

8.2.5 tr_hmac_key_send()

The TR calculates a 160-bit HMAC key for the UDCP to use. During initialization, the TR calculates the key, encrypts it with the UDCP's public key, and sends it to the UDCP. The TR uses this HMAC key to verify the data integrity as well as to authenticate the resolve tuning request from the UDCP.

The TR SHALL send a new HMAC key to the UDCP after any of the following conditions:

An tr_init_req() APDU is received from the TR.

The TR receives a resolve tuning request from the UDCP that does not authenticate with the current HMAC key.

The TR MAY send a new HMAC key to the UDCP after any of the following conditions:

The TR receives a resolve tuning request from the UDCP that contains a duplicate request_id.

The TR determines it is necessary for security reasons.

In the case that the TR determines that updating the HMAC key is necessary for security reasons, the TR SHALL NOT send a new HMAC key to the UDCP more than once per hour.

The TR SHALL calculate a random 160-bit HMAC key to send in the tr_hmac_key_send() APDU.

The TR SHALL encrypt the HMAC key with the UDCP's RSA Public Key (extracted from inside the Host certificate) using the PKCS1-V1_5 scheme, block type 0x02. After encryption, the output will be of length 128 octets (1024 bits).

If the TR authenticates the UDCP, the TR SHALL send the encrypted HMAC key in the tr_hmac_key_send() APDU as defined in Table 7, within five seconds of receiving the challenge_rsp() APDU.

The UDCP MAY conclude that authentication has failed if it does not receive a tr_hmac_key_send() APDU within five seconds.

The UDCP SHALL use the tr_hmac_key sent by the TR as the HMAC key.

The TR can send a new HMAC key under certain conditions. If the UDCP receives a new HMAC key, it SHALL use the most recent HMAC key to calculate the resolve_tuning_digest in all subsequent resolve_tuning_req APDUs.

Table 7 - tr_hmac_key_send APDU

Syntax	# of bits	Mnemonic
tr_hmac_key_send(){		
tr_hmac_key_send_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
tr_hmac_key_encrypted	1024	uimsbf
}		

tr_hmac_key_send_tag 0x010A

trif_revision_code 0x01

tr_hmac_key_encrypted The HMAC Key encrypted using UDCP's Public Key.

8.2.6 channel_table_req()

The UDCP SHALL send a channel_table_req() APDU as defined in Table 8 to request a channel table anytime after receiving the tr_init_rsp() APDU.

Table 8 - Channel Table Request APDU

Syntax	# of bits	Mnemonic
channel_table_req(){		
channel_table_req_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
}		

channel_table_req_tag 0x0107

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

8.2.7 channel_table_rsp()

The TR SHALL send the channel_table_rsp() APDU as defined in Table 9 within five seconds after receiving a channel_table_req() APDU.

If the TR sends the channel_table_rsp() APDU with a table_status other than table available (0x00), the TR SHALL send the channel_table_update APDU as soon as it becomes available.

Table 9 - Channel Table Response APDU

Syntax	# of bits	Mnemonic
channel_table_rsp(){		
channel_table_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
table_status	8	uimsbf
if (table_status == 0x00) {		
trif_channel_table()		
}		
}		

channel_table_rsp_tag 0x0108

trif_revision_code 0x01

request_id The value in the channel_table_req() APDU

table_status The following values are used to define the table status:

0x00 – Table available
0x01 – Table not available (no reverse path established)
0x02 – Table not available (not received)
0x03 – Table not available (other reason)
0x04-0xFF – Reserved

trif_channel_table() As defined in section 8.6.1.

8.2.8 channel_table_update()

The TR SHALL send the channel_table_update() APDU as defined in Table 10 unsolicited if there is any change detected within the channel table.

Table 10 - Channel Table Update Response APDU

Syntax	# of bits	Mnemonic
channel_table_update(){		
channel_table_update_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
reserved	24	'0xFFFFFF'
trif_channel_table()		
}		

channel_table_update_tag 0x0109

trif_revision_code 0x01

trif_channel_table() As defined in section 8.6.1.

8.3 Resolve Tuning Messages

The UDCP uses the Resolve Tuning Messages for retrieving information for tuning any channel or source ID. It is expected that resolving the source ID tuning information is only used when an [SCTE 18] EAS message is received.

Table 11 - Resolve Tuning Messages (Informative)

Message Name	Message Tag (hex)	Direction	Description
resolve_tuning_req	02 01	UDCP → TR	UDCP sends resolve tuning to TR.
resolve_tuning_rsp	02 02	UDCP ← TR	TR response to resolve tuning.
resolve_tuning_update	02 03	UDCP ← TR	TR update to resolve tuning.
resolve_tuning_cnf	0204	UDCP → TR	UDCP sends resolve tuning confirmed

8.3.1 resolve_tuning_req()

The UDCP SHALL send the resolve_tuning_req() APDU as defined in Table 12 at any time after an HMAC key is delivered by the TR to request tuning information from the TR.

Upon receiving an [SCTE 18] message from the CableCARD, the UDCP SHALL use the resolve_tuning_req() APDU to obtain the tuning values of the source ID defined in the [SCTE 18] message.

Table 12 - Resolve Tuning Request APDU

Syntax	# of bits	Mnemonic
resolve_tuning_req(){		
resolve_tuning_req_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
ltsid	8	uimsbf
channel_source_type	1	uimsbf
reserved	4	'1111'
tuner_use_status	3	uimsbf
if (channel_source_type == 0b0) {		
reserved	7	'1111111'
channel_number	17	uimsbf
} else {		
reserved	8	'11111111'
source_id	16	uimsbf
}		
resolve_tuning_digest	160	uimsbf
}		

resolve_tuning_req_tag 0x0201

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

ltsid Local transport stream ID as assigned to tuner in the discovery process (tr_init_req() APDU).

channel_source_type Selects between a channel or source ID

0b0 – Channel Number
0b1 – Source ID

tuner_use_status Type of tune request.

0x0 – Live full screen video (no HDD recording)
0x1 – Live full screen video (HDD recording)
0x2 – Live PIP or POP video (no HDD recording)
0x3 – Live PIP or POP video (HDD recording)
0x4 – Recording only
0x5 – Inactive (**Note:** this value is only applicable to udcp_status_rsp() and udcp_status_update() APDUs)
0x6 – Speculative Recording
0x7 – Reserved

channel_number Channel number to be tuned.

source_id Source ID as defined in [SCTE 65].

resolve_tuning_digest The previous fields in the body (not including the tag and length fields) of the APDU are combined, and HMAC-SHA1 performed using the HMAC Key as described in [RFC 2104]. For a block size of 64 Bytes, and hash function SHA1 output size 20 Bytes, the key (20 Bytes) needs to be padded as described in the RFC with ipads/opads to make it the same size as block size.

The UDCP SHALL NOT send a resolve_tuning_req() APDU with the tuner_use_status = 0x5 (inactive).

The UDCP SHALL increment the request_id counter for every transmission of the resolve_tuning_req() APDU.

If the resolve_tuning_digest is not authenticated with the HMAC key, the TR SHALL send a new key using the tr_hmac_key_send() APDU.

In the resolve_tuning_req() APDU, the UDCP SHALL only report the tuner_use_status values associated with HDD Recording (0x1, 0x3 or 0x4) when a user-initiated, scheduled or timed recording is in effect that will always be followed by an udcp_status_update() APDU that reports 0x0, 0x2, or 0x5, 0x6 at the end of the recording.

In the resolve_tuning_req() APDU, the UDCP SHALL report tuner_use_status values associated with opportunistic or speculative recording that was not user-initiated as 0x6.

In the resolve_tuning_req() APDU, the UDCP SHALL report tuner_use_status values associated with a non-displayed time-shift-buffer as 0x6.

In the resolve_tuning_req() APDU, the UDCP SHALL report tuner_use_status values associated with live-off-disk on-screen viewing or viewing from the time-shift-buffer on DVR products as 0x0 or 0x2.

8.3.2 resolve_tuning_rsp()

The TR SHALL send the resolve_tuning_rsp() APDU as defined in Table 13 within five seconds after receiving a resolve_tuning_req() APDU.

Table 13 - Resolve Tuning Response APDU

Syntax	# of bits	Mnemonic
resolve_tuning_rsp(){		
resolve_tuning_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
ltsid	8	uimsbf
reserved	7	'111111111'
channel_number	17	uimsbf
tune_status	8	uimsbf
tune_frequency	16	uimsbf
tune_program_number	16	uimsbf
tune_source_id	16	uimsbf
tune_transmission_system	4	uimsbf
tune_inner_coding	4	uimsbf
tune_split_bitstream_mode	1	bslbf
zero	2	'00'
tune_modulation_format	5	uimsbf
zero	4	'0000'
tune_symbol_rate	28	uimsbf
}		

resolve_tuning_rsp_tag 0x0202

trif_revision_code 0x01

request_id The value in the resolve_tuning_req() APDU.

ltsid Local transport stream ID as assigned to tuner in the discovery process (udcp_profile_rsp() APDU)

channel_number Channel to be tuned. If the channel_source_type in the associated resolve_tuning_req() APDU is 0b1, then this value is set to 0x00000 by the TR and is ignored by the UDCP.

tune_status	Tuning status 0x00 – Tuning resolution successful 0x01 – Tuning resolution cannot be determined at this time (Channel not available) 0x02 – Tuning resolution cannot be determined at this time (Network not available) 0x03 – Tuning information not available (channel or source not defined in table) 0x04 – Illegal channel number (i.e., >99,999) 0x05 – Invalid LTSID (LTSID greater than number of tuners) 0x06 – Invalid resolve_tuning_digest 0x07 – Source ID parameters not known 0x08 – Invalid tuner_use_status 0x09-0xFF - Reserved
tune_frequency	If tune_status == 0x00, contains the frequency for the Host to tune (Center frequency). The frequency is calculated by multiplying tune_frequency by 0x0.05 MHz (50 kHz resolution). If channel_status ≠ 0x00, then 0x0000.
tune_program_number	If tune_status == 0x00 AND tune_modulation_format ≠ 0x00, contains the MPEG-2 program number, else 0x0000.
tune_source_id	If tune_status == 0x00, contains the [SCTE 65] source ID (used in CableCARD ca_pmt() APDU, see [CCIF2.0]), else 0x0000. If resolve_tuning_req() APDU request_type == 0b1, contains the source_id value in resolve_tuning.
tune_transmission_system	If tune_status == 0x00, value as defined in table 5.7 of [SCTE 65], else 0x0.
tune_inner_coding	If tune_status == 0x00, value as defined in table 5.8 of [SCTE 65], else 0x0.
tune_split_bitstream_mode	If tune_status = 0x00, value as defined in [SCTE 65], else 0b0.
tune_modulation_format	Value as defined in table 5.9 of [SCTE 65] with the clarification that 0x00 means NTSC.
tune_symbol_rate	If tune_status == 0x00, value as defined in [SCTE 65], else 0x00000000.

If the UDCP receives a resolve_tuning_rsp() APDU with tune_status of 0x01, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_rsp() APDU with tune_status of 0x01, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is displaying an EAS message.

If the UDCP receives a resolve_tuning_rsp() APDU with tune_status of 0x02, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_rsp() APDU with tune_status of 0x02, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is displaying an EAS message.

Section 10.3 contains an example of the message to be displayed in these cases.

8.3.3 resolve_tuning_update()

The TR SHALL send the resolve_tuning_update() APDU as defined in Table 14 if there is any change to the tuning information for each ltsid for which a resolve_tuning_req() APDU has been received.

Table 14 - Resolve Tuning Update APDU

Syntax	# of bits	Mnemonic
resolve_tuning_update(){		
resolve_tuning_update_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
ltsid	8	uimsbf
reserved	7	'11111111'
channel_number	17	uimsbf
tune_status	8	uimsbf
tune_frequency	16	uimsbf
tune_program_number	16	uimsbf
channel_source_id	16	uimsbf
tune_transmission_system	4	uimsbf
tune_inner_coding	4	uimsbf
tune_split_bitstream_mode	1	bslbf
zero	2	'00'
tune_modulation_format	5	uimsbf
zero	4	'0000'
tune_symbol_rate	28	uimsbf
}		

resolve_tuning_update_tag 0x0203

trif_revision_code 0x01

request_id The request_id from the most recent resolve_tuning_req() APDU for the associated ltsid.

ltsid Local transport stream ID as assigned to tuner in the discovery process (udcp_profile_rsp() APDU).

channel_number Channel to be tuned. If the channel_source_type in the associated resolve_tuning_req() APDU is 0b1, then this value is set to 0x00000 by the TR and is ignored by the UDCP.

tune_status
- 0x00 – Tuning resolution successful
- 0x01 – Tuning resolution cannot be determined at this time (Channel not available)
- 0x02 – Tuning resolution cannot be determined at this time (Network not available)
- 0x03 – Tuning information not available (channel or source not defined in table)
- 0x04 – Illegal channel number (i.e. >99,999)
- 0x05 – Invalid LTSID (LTSID greater than number of tuners
- 0x06 – Invalid resolve_tuning_signature
- 0x07 – Source ID parameters not known
- 0x08-0xFF - Reserved

tune_frequency If tune_status == 0x00, contains the frequency for the Host to tune (Center frequency for 64 and 256 QAM, video frequency for NTSC). The frequency is calculated by multiplying tune_frequency by 0x0.05 MHz (50 kHz resolution). If channel_status ≠ 0x00, then 0x0000.

tune_program_number If tune_status == 0x00 AND tuning_modulation ≠ 0x00, contains the MPEG-2 program number, else 0x0000.

tune_source_id	If tune_status == 0x00, contains the [SCTE 65] source ID (used in CableCARD ca_pmt() APDU, see [CCIF2.0]), else 0x0000. If resolve_tuning_req() APDU request_type == 0b1, contains the source_id value in resolve_tuning.
tune_transmission_system	If tune_status == 0x00, value as defined in table 5.7 of [SCTE 65], else 0x0.
tune_inner_coding_mode	If tune_status == 0x00, value as defined in table 5.8 of [SCTE 65], else 0x0.
tune_split_bitstream_mode	If tune_status = 0x00, value as defined in [SCTE 65], else 0b0.
tune_modulation_format	Value as defined in table 5.9 of [SCTE 65] with the clarification that 0x00 means NTSC.
tune_symbol_rate	If tuning_status == 0x00, value as defined in [SCTE 65], else 0x0000000.

The TR SHALL include in the request_id field the most recent request_id received in a resolve_tuning_req() APDU from the UDCP for the associated ltsid.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x01, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x01, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is displaying an EAS message.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x02, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x02, then the UDCP SHALL create a visible message to the user indicating the channel is temporarily unavailable, unless the UDCP is displaying an EAS message.

Section 10.3 contains an example of the message to be displayed in these cases.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and the channel_number has changed from the previous resolve_tuning_rsp() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and the channel_number has changed from the previous resolve_tuning_rsp() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is displaying an EAS message.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and the channel_number has changed from the previous resolve_tuning_update() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and the channel_number has changed from the previous resolve_tuning_update() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is displaying an EAS message.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and source_id has changed from the previous resolve_tuning_rsp() with the same request_id, then the UDCP SHALL create a visible message to the

user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and source_id has changed from the previous resolve_tuning_rsp() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is displaying an EAS message.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and source_id has changed from the previous resolve_tuning_update() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is in standby.

If the UDCP receives a resolve_tuning_update() APDU with a tune_status of 0x00 and source_id has changed from the previous resolve_tuning_update() with the same request_id, then the UDCP SHALL create a visible message to the user indicating the requested channel is temporarily unavailable and an alternative is available, unless the UDCP is displaying an EAS message.

Section 10.1 contains an example of the message to be displayed in these cases.

8.3.4 resolve_tuning_cnf()

The UDCP SHALL send the resolve_tuning_cnf() APDU as defined in Table 15 after receiving either the resolve_tuning_rsp() OR resolve_tuning_update() APDUs AND the UDCP has successfully tuned to the channel. The period of time that the UDCP defines to be unable to lock is outside the scope of this specification and is defined by the UDCP vendor.

Table 15 - Resolve Tuning Confirm APDU

Syntax	# of bits	Mnemonic
resolve_tuning_cnf(){		
resolve_tuning_cnf_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
udcp_status()	8	uimsbf
}		

resolve_tuning_cnf_tag 0x0204

trif_revision_code 0x01

udcp_status() As defined in 8.6.5

The UDCP SHALL NOT send a resolve_tuning_cnf() APDU with the tuner_use_status = 0x5 (inactive).

8.4 Status Messages

Table 16 - Status Messages (Informative)

Message Name	Message Tag (hex)	Direction	Description
tr_status_req	03 01	UDCP → TR	UDCP requests TR status
tr_status_rsp	03 02	UDCP ← TR	TR sends status to UDCP
udcp_status_req	03 03	UDCP ← TR	TR requests UDCP status
udcp_status_rsp	03 04	UDCP → TR	UDCP sends status to TR
tr_status_update	03 05	UDCP ← TR	TR sends updated status to UDCP
udcp_status_update	03 06	UDCP → TR	UDCP sends updated status to TR
tr_message	03 07	UDCP ← TR	TR sends message for UDCP to display
tr_message_rsp	03 08	UDCP → TR	UDCP response to tr_message

8.4.1 tr_status_req()

The UDCP SHALL send the tr_status_req() APDU as defined in Table 17 to request the status of the TR.

Anytime after receiving the tr_init_rsp() APDU, the UDCP MAY send a tr_status_req() APDU.

The UDCP SHALL NOT send tr_status_req() APDUs with a periodicity greater than one every five seconds. **Note:** The UDCP may use this APDU to poll the TR for health and status, and therefore the polling needs to be restricted so that it doesn't overwhelm the TR.

Table 17 - TR Status Request APDU

Syntax	# of bits	Mnemonic
`tr_status_req(){`		
`  tr_status_req_tag`	16	uimsbf
`  length_field`	16	uimsbf
`  trif_revision_code`	8	uimsbf
`  request_id`	16	uimsbf
`}`		

tr_status_req_tag 0x0301

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

8.4.2 tr_status_rsp()

The TR SHALL send a tr_status_rsp() APDU as defined in Table 18 within five seconds of receiving a tr_status_req() APDU, except when a TR firmware upgrade is in progress.

Table 18 - TR Status Response APDU

Syntax	# of bits	Mnemonic
tr_status_rsp(){		
tr_status_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
tr_status()		
}		

tr_status_rsp_tag 0x0302

trif_revision_code 0x01

request_id The value in the tr_status_req() APDU.

tr_status() As defined in section 8.6.4.

8.4.3 udcp_status_req()

The TR SHALL send a udcp_status_req() APDU as defined in Table 19 to the UDCP to request status of the UDCP.

The TR SHALL NOT send udcp_status_req() APDUs with a periodicity greater than one every five seconds. **Note:** The TR may use this APDU to poll the UDCP for health and status, and therefore the polling needs to be restricted so that it doesn't overwhelm the UDCP.

Table 19 - UDCP Status Request APDU

Syntax	# of bits	Mnemonic
udcp_status_req(){		
udcp_status_req_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
}		

udcp_status_req_tag 0x0303

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

8.4.4 udcp_status_rsp()

The UDCP SHALL send a udcp_status_rsp() APDU as defined in Table 20 within five seconds of receiving a udcp_status_req() APDU, except when a UDCP is unable to respond, such as during a firmware upgrade.

Table 20 - UDCP Status Response APDU

Syntax	# of bits	Mnemonic
udcp_status_rsp(){		
udcp_status_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
udcp_status()		
}		

udcp_status_rsp_tag 0x0304

trif_revision_code 0x01

request_id The value in the udcp_status_req() APDU.

udcp_status() As defined in 8.6.5

8.4.5 tr_status_update()

The TR SHALL send the tr_status_update() APDU as defined in Table 21 only to inform the UDCP of a change in the status of the TR.

Table 21 - TR Status Update APDU

Syntax	# of bits	Mnemonic
tr_status_update(){		
tr_status_update_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
reserved	16	'0xFFFF'
tr_status()		
}		

tr_status_update_tag 0x0305

trif_revision_code 0x01

tr_status() As defined in section 8.6.4.

8.4.6 udcp_status_update()

The UDCP SHALL send the udcp_status_update() APDU as defined in Table 22 only to inform the TR of a change in the status of the UDCP.

Table 22 - UDCP Status Update APDU

Syntax	# of bits	Mnemonic
udcp_status_update(){		
udcp_status_update_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
reserved	16	'0xFFFF'
udcp_status()		
}		

udcp_status_update_tag 0x0306

trif_revision_code 0x01

udcp_status() As defined in 8.6.5

The UDCP SHALL send the udcp_status_update() after the channel table has been completely received after initialization.

8.4.7 tr_message()

The TR SHALL use the tr_message() APDU as defined in Table 23 to transmit a message to the UDCP (see Section 10 for examples).

Table 23 - TR MessageAPDU

Syntax	# of bits	Mnemonic
tr_message(){		
tr_message_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	
ltsid	8	uimsbf
message_type	8	uimsbf
message_length	16	uimsbf
for (i=0; i<message_length; i++){		
message_byte	8	uimsbf
}		
}		

tr_message_tag 0x0307

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

ltsid Local transport stream ID as assigned to tuner in the discovery process (udcp_profile_rsp() APDU).

message_type Type of message

0x00 User Inactivity Message
0x01 Unspecified Message
0x02-0xFF Reserved

message_length Number of bytes in the message.

message_byte Message byte.

The TR Message message_byte data sent by the TR SHALL be formatted as defined in Annex A of [CCIF2.0].

If the message_type >= 0x01, then the UDCP SHALL display the message in the APDU if the UDCP is not in standby.

If the message_type >= 0x01, then the UDCP SHALL display the message in the APDU if the UDCP is not displaying an EAS message.

If the message_type < 0x01, then the UDCP SHALL either display the message in the APDU or an equivalent message for the message type if the UDCP is not in standby.

If the message_type < 0x01, then the UDCP SHALL either display the message in the APDU or an equivalent message for the message type if the UDCP is not displaying an EAS message.

Note: It is expected that the operator will not use message_type 0x01 – Unspecified Message for promotion, for advertising, or other messaging not related to switched video.

If the UDCP chooses not to display the message in the APDU, the UDCP SHALL display an equivalent message. For example, if the UDCP is configured for a different language, then a message in that language is displayed.

If the UDCP receives message_type 0x00 "User Inactivity Message" within five seconds of having sent a udcp_status_update() or udcp_status_rsp() APDU to the TR with 'user_activity_detected = true', or within 10 minutes of having responded affirmatively to a previous message_type 0x00, based on human activity as defined in 8.6.5, then the UDCP MAY resend a udcp_status_update() APDU with 'user_activity_detected = true' without re-presenting the inactivity message to the screen.

8.4.8 tr_message_rsp

The UDCP SHALL send the tr_message_rsp() as defined in Table 24 within five seconds of receiving a tr_message() APDU, except when a UDCP firmware upgrade is in progress.

Table 24 - TR Message Response APDU

Syntax	# of bits	Mnemonic
tr_message_rsp(){		
tr_message_rsp_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
message_response_code	8	uimsbf
}		

tr_message_rsp_tag 0x0308

trif_revision_code 0x01

request_id The value in the tr_message() APDU.

message_response_code TR Message Response Code

0x00 – Message displayed
0x01 – Message not displayed, UDCP in standby
0x02 – Message not displayed, not in video display mode
0x03 – Message not displayed, no program present
0x04 – Message not displayed, undefined reason
0x05 – Message not displayed, UDCP displaying EAS message
0x06-0xF – Reserved

8.5 Diagnostic Messages

Table 25 - Diagnostic Messages(Informative)

Message Name	Message Tag (hex)	Direction	Description
tr_diag_req	04 01	UDCP → TR	UDCP requests TR diagnostics
tr_diag_rsp	04 02	UDCP ← TR	TR sends diagnostics to UDCP

8.5.1 tr_diag_req()

The UDCP SHALL send the tr_diag_req() APDU as defined in Table 26 to request TR diagnostics.

Table 26 - TR Diagnostic Request APDU

Syntax	# of bits	Mnemonic
tr_diag_req(){		
tr_diag_req_tag	16	uimsbf
length_field	16	uimsbf
trif_revision_code	8	uimsbf
request_id	16	uimsbf
url_length	16	uimsbf
for (i=0; i<url_length; i++) {		
url_byte	8	uimsbf
}		
}		

tr_diag_req_tag 0x0401

trif_revision_code 0x01

request_id A modulo 65,536 counter that increments for each request APDU.

url_length The number of URL bytes in the following for loop. A value of 0x0000 represents a request for the root page of diagnostic data provided by the TR.

url_byte Each url_byte is an octet of a parameter that defines the location of the requested page of diagnostic data.

The URL of the root page of diagnostic data provided by the TR SHALL be "tr:///diag/default.html".

When requesting the root page of diagnostic data provided by the TR, the UDCP MAY optionally specify a url_length of 0x0000 without specifying the actual locator in the url_byte loop that follows.

The UDCP SHALL only request URLs that were obtained from links found in MMI pages originating from a prior tr_diag_rsp() message.

8.5.2 tr_diag_rsp()

The TR SHALL send the tr_diag_rsp() APDU as defined in Table 27 within five seconds of receiving a tr_diag_req() APDU, except when a TR firmware upgrade is in progress.

The UDCP SHALL provide a mechanism to request and view the TR diagnostics that are received from the TR diagnostic query. If the user requests to view the TR diagnostics through this mechanism, the UDCP SHALL issue

a tr_diag_req(). Upon receipt of the tr_diag_rsp() APDU from the TR, the UDCP SHALL display the TR diagnostics that are received from the TR diagnostic query, unless the user cancels the request by exiting the mechanism.

The UDCP SHALL NOT display the TR diagnostics if the request_id of the tr_diag_rsp() APDU does not equal the request_id of a previously transmitted tr_diag_req() APDU.

The UDCP SHALL NOT display the TR diagnostics if the UDCP is in standby.

The UDCP SHALL NOT display the TR diagnostics if an EAS message is being displayed by the UDCP.

The TR SHALL only issue a tr_diag_rsp() APDU to the UDCP in response to a tr_diag_req() APDU from the UDCP.

Table 27 - TR Diagnostic Response APDU

Syntax	# of bits	Mnemonic
`tr_diag_rsp(){`		
`  tr_diag_rsp_tag`	`16`	`uimsbf`
`  length_field`	`16`	`uimsbf`
`  trif_revision_code`	`8`	`uimsbf`
`  request_id`	`16`	`uimsbf`
`  tr_diagnostics()`		
`}`		

tr_diag_rsp_tag 0x0402

trif_revision_code 0x01

request_id The value in the tr_diag_req() APDU.

tr_diagnostics() As defined in section 8.6.6.

8.6 TR Interface Functions

8.6.1 trif_channel_table()

Table 28 - trif_channel_table()

Syntax	No. of Bits	Mnemonic
trif_channel_table() {		
table_length	16	uimsbf
trif_table_revision	8	uimsbf
version_number	8	uimsbf
total_number_of_blocks	16	uimsbf
reserved	7	'1111111'
total_number_of_defined_channels	17	uimsbf
block_number	16	uimsbf
number_of_channels	16	uimsbf
for (i=0; i<number_of_channels; i++) {		
channel_type	4	uimsbf
reserved	3	'111'
channel_number	17	uimsbf
short_name	7*16	uimsbf
}		
}		

table_length Number of bytes following in this table.

trif_table_revision Version of the table format, currently set to 0x01. **Note:** All subsequent revisions are to be made by adding parameters at the end of the previous revisions.

version_number Current version of the data in the entire table, not just this block. This is a modulo 256 count value that is incremented whenever any data changes.

total_number_of_blocks Number of blocks that will be transmitted. Each block is carried in a separate APDU.

total_number_of_defined_channels Total number of defined channels available to the UDCP.

block_number Block number of this portion of the channel table.

number_of_channels Number of channels in the following loop.

channel_type As defined in section 5.3.2 of [SCTE 65].

channel_number Channel number.

short_name Name of the channel, up to seven unicode characters.

The TR SHALL limit the number of records in the trif_channel_table to 65,535.

If the short_name of the channel is less than seven characters, the TR SHALL set the remaining characters to null.

The TR SHALL send the channel table records in sequential order.

The TR SHALL begin the block numbering in the transmission of the channel table with block number zero as the first block.

8.6.2 udcp_profile()

Table 29 - udcp_profile()

Syntax	No. of Bits	Mnemonic
udcp_profile() {		
function_length	16	uimsbf
udcp_profile_revision	8	uimsbf
number_of_tuners	8	uimsbf
number_of_video_codecs	8	uimsbf
for (i=1; i<number_of_video_codecs; i++){		
video_codec	8	uimsbf
}		
number_of_audio_codecs	8	uimsbf
for (i=1; i<number_of_audio_codecs; i++){		
audio_codec	8	uimsbf
}		
upper_frequency_tuning_range	16	uimsbf
manufacturer_id	24	uimsbf
hardware_version_num	16	uimsbf
number_of_str_bytes	8	uimsbf
for (i=0; i<number_of_str_bytes; i++){		
str_byte	8	uimsbf
}		
}		

function_length Number of bytes following in this function.

udcp_profile_revision Version of the table format, currently set to 0x01. **Note:** All subsequent revisions are backwards compatible in that any additional parameters are added at the end of this profile.

number_of_tuners Number of tuners in the UDCP. **Note:** Each tuner in the UDCP is identified by an LTSID by the UDCP. The association between a tuner and LTSID does not change after reboot.

number_of_video_codecs Number of video codecs in the UDCP.

video_codec Video codec supported

0x00 – NTSC
0x01 – MPEG-2
0x02 – MPEG-4
0x04 – Undefined
0x05-0xff – Reserved

number_of_audio_codecs Number of audio codecs in the UDCP.

audio_codec Audio codec supported

0x00 – AC-3
0x01 – MPEG-2, 2 channel
0x02 – MPEG-1, Layer 1
0x03 – MPEG-1, Layer 2
0x04 – MPEG-1, Layer 3
0x05 – Dolby™ Digital Plus (AC-3 Plus)
0x06 – AAC
0x07 – AAC-HE
0x08 - Undefined
0x09-0xFF - Reserved

upper_frequency_tuning_range Upper band edge of the highest tunable QAM channel. The frequency is calculated by multiplying tune_frequency by 0x0.05 MHz (50 kHz resolution).

manufacturer_id UDCP manufacturer's IEEE assigned Organization Unique Identifier (OUI).

hardware_version_num The TR hardware version number expressed as a single integer.

number_of_str_bytes The number of ASCII string bytes to follow.

str_byte String of 7-bit ASCII characters that the UDCP manufacturer provides to aid in uniquely identifying the UDCP software version(s) for diagnostic purposes.

The UDCP SHALL utilize a unique enumerated LTSID (local transport stream identifier) for each tuner in the UDCP.

The LTSID is zero based, so the first tuner is ltsid 0x00.

The UDCP SHALL NOT change the relationship between the LTSID and the tuner after reboot.

If the TR receives an LTSID greater than the number of tuners in udcp_profile(), the TR SHALL report this as an error condition. For instance, if the UDCP reported the value 1 for number_of_tuners in its udcp_profile() and later sent a resolve_tuning_req() APDU specifying LTSID 0x01, the TR would respond by sending a resolve_tuning_rsp() APDU with tune_status = 0x05 (invalid LTSID).

The UDCP SHALL provide a unique hardware_version_num other than zero.

The UDCP SHALL provide unique software version(s) information as a string of ASCII text.

8.6.3 tr_profile()

Table 30 - tr_profile ()

Syntax	No. of Bits	Mnemonic
tr_profile() {		
function_length	16	uimsbf
tr_profile_revision	8	uimsbf
number_of_tuners	8	uimsbf
manufacturer_id	24	uimsbf
hardware_version_num	16	uimsbf
number_of_str_bytes	8	uimsbf
for (i=0; i<number_of_str_bytes; i++){		
str_byte	8	uimsbf
}		
}		

function_length Number of bytes following in this function.

tr_profile_revision Version of the table format, currently set to 0x01 **Note:** All subsequent revisions are backwards compatible in that any additional parameters are added at the end of this profile.

number_of_tuners Number of tuners that the TR supports.

manufacturer_id TR manufacturer's IEEE assigned Organization Unique Identifier (OUI).

hardware_version_num The TR hardware version number expressed as a single integer.

number_of_str_bytes The number of ASCII string bytes to follow.

str_byte A string of 7-bit ASCII characters that the TR manufacturer provides to aid in uniquely identifying the tuning resolver software versions(s) for diagnostic purposes.

The TR SHALL support at least two tuners.

Note: The UDCP may have more or fewer tuners than the TR supports.

If the TR receives an LTSID greater than the number of tuners in tr_profile, the TR SHALL report this as an error condition.

The TR SHALL provide a hardware version number other than zero.

The TR SHALL provide unique software version(s) information as a string of ASCII text.

If the UDCP presents the contents of the manufacturer_id to the user (e.g., via presentation of an on screen display), it is recommended that the UDCP display the Manufacturer's ID in hexadecimal notation as 'xx-xx-xx'.

8.6.4 tr_status

Table 31 - tr_status

Syntax	No. of Bits	Mnemonic
tr_status() {		
function_length	16	uimsbf
tr_status_revision	8	uimsbf
version_number	8	uimsbf
downstream_status	8	uimsbf
upstream_status	8	uimsbf
authentication_status	8	uimsbf
tr_operational_status	8	uimsbf
max_upgrade_time	8	uimsbf
number_of_tuners	8	uimsbf
}		

function_length Number of bytes following in this function.

tr_status_revision Version of the table format, currently set to 0x01. **Note:** All subsequent revisions are backwards compatible in that any additional parameters are added at the end of this profile.

version_number A modulo 256 counter that increments if the status in this APDU is different from the last status transmitted.

downstream_status The downstream status of the TR.

0x00 –Downstream RF locked and receiving valid downstream messages
0x01 –Downstream RF not locked
0x02 –Downstream RF locked waiting for downstream messages
0x03-0xFF - Reserved

upstream_status The upstream status of the TR.

0x00 –Upstream connection complete
0x01 –Upstream not connecting
0x02 –Waiting for connection complete
0x03 – TR disabled
0x04-0xFF – Reserved

authentication_status The authentication status of the TR/UDCP.

0x00 – Authenticated
0x01 – Authentication in progress
0x02 – Not authenticated
0x03-0xFF – Reserved

tr_operational_status The current operational status of the TR.

0x00 – TR ready to resolve
0x01 – TR initializing
0x02 – TR upgrade in progress
0x03 – TR disabled
0x04 – TR unavailable (unspecified reason)
0x05-0xFF – Reserved

max_upgrade_time When the tr_operational_status = 0x02, the maximum time in minutes to perform the TR firmware upgrade. When this field is updated it shows remaining time.

number_of_tuners Number of tuners CURRENTLY supported by the TR.

If the TR sends the tr_operational_status != 0x00 in the tr_status() interface function, the UDCP SHALL discontinue use of the channel_table and start utilizing the information carried in [SCTE 65].

The TR SHALL NOT change the number of tuners it supports except after a USB reset or once a second.

Note: The number of UDCP tuners currently supported by the TR may change depending on network load conditions, subscription levels or other conditions.

The number of UDCP tuners supported by the TR SHALL remain greater than or equal to the minimum number of tuners it must support as defined in Section 8.6.3 in the tr_profile() APDU.

Once the TR has sent a tr_status APDU with tr_operational_state set to a value other than 0x01 'initializing', it SHALL NOT send a tr_status APDU with tr_operational_status = 0x01 unless it resets or is sent a tr_init_req APDU from the UDCP.

The TR SHALL send a tr_status_update() APDU with tr_operational_status equal to 0x00 - 'TR ready to resolve' when it is fully prepared to process a resolve_tuning_req() APDU from the UDCP for any valid channel_source_type.

The UDCP SHALL NOT send a resolve_tuning_req() APDU to the TR until the TR sends a tr_status_update() APDU with tr_operational_status of 0x00.

8.6.5 udcp_status

Table 32 - udcp_status

Syntax	No. of Bits	Mnemonic
udcp_status() {		
function_length	16	uimsbf
udcp_status_revision	8	uimsbf
version_number	8	uimsbf
user_activity_detected	1	bslbf
reserved	7	'1111111'
udcp_status	8	uimsbf
number_of_tuners	8	uimsbf
for (i=0; i<number_of_tuners; i++){		
tuner_status	4	uimsbf
tuner_use_status	3	uimsbf
tuner_channel_number	17	uimsbf
tuner_source_id	16	uimsbf
ca_allowed_status	1	bslbf
reserved	7	'1111111'
}		
}		

function_length Number of bytes following in this function.

udcp_status_revision Version of the table format, currently set to 0x01. **Note:** All subsequent revisions are backwards compatible in that any additional parameters are added at the end of this profile.

version_number A modulo 256 counter that increments if the status in this APDU is different from the last status transmitted.

user_activity_detected Boolean flag set when there has been user activity since the last udcp_status() was transmitted.

0b1 – User activity since last udcp_status()
0b0 – No user activity since last udcp_status()

udcp_status Status of the UDCP.

0x00 – UDCP in standby
0x01 – UDCP in on state
0x02-0xFF - Reserved

number_of_tuners Number of tuners in the UDCP. The entries in the tuner loop following this field in the APDU are ordered by ascending LTSID starting at LTSID 0.

tuner_status Status of tuner.

0x0 – Tuner active
0x1 – Tuner inactive
0x2 – Tuning in progress
0x3 – Tuning failed (not used when ca_allowed_status = 0b0)
0x4-0xF – Reserved

tuner_use_status Current type of tune request. Note that this is intended to convey the current purpose of an associated resolve_tuning_req(), as opposed to conveying tuner status changes.

0x0 – Live full screen video (no HDD recording)
0x1 – Live full screen video (HDD recording)
0x2 – Live PIP or POP video (no HDD recording)
0x3 – Live PIP or POP video (HDD recording)

	0x4 – User Directed Recording only 0x5 – Inactive 0x6 – Speculative Recording 0x7 – Reserved
tuner_channel_number	If tuner_status == 0x0, current channel number of tuner i.
tuner_source_id	If tuner_status == 0x0, current source ID of tuner i.
ca_allowed_status	CableCARD has allowed tuning of this program. 0b1 – CA allowed or CA status pending) 0b0 – CA not allowed (when reported by card)

If the tuner_use_status is not 'inactive', the UDCP SHALL set user_activity_detected and transmit the udcp_status_update() APDU within one second of when human activity is detected. **Note:** Multiple detections of human activity within one second are expected to be contained within one udcp_status().

The UDCP MAY set the user_activity_detected_flag if the tuner_use_status is 'inactive'.

The UDCP SHALL define human activity as when either a remote control code is received OR a front panel key is pressed, AND cable content is being displayed, recorded, or sent to any digital or analog outputs.

When the UDCP tunes, it may not immediately have ca_allowed_status from the CableCARD. In cases where the UDCP does not have ca_allowed_status, it may report ca_allowed_status =0b1. As the UDCP receives updates of the ca_allowed_status from the CableCARD, it SHALL send udcp_status_update() APDUs to the TR to reflect this updated status.

The UDCP SHALL send a udcp_status_update() APDU with the tuner_use_status = 0x5 (inactive) for a given LTSID when the associated tuner output is changed such that it is not being rendered, recorded, or transmitted outside of the UDCP.

The UDCP SHALL NOT use the tuning parameters obtained from a prior resolve_tuning_rsp() or resolve_tuning_update() APDU for rendering, recording, or transmitting further output from the associated tuner tuned according to those parameters after sending a udcp_status_update() with the tuner_use_status = 0x5 (inactive) for the corresponding LTSID.

The TR SHALL NOT send resolve_tuning_update() APDUs associated with a prior resolve_tuning_req() APDU to the UDCP after having received a udcp_status_update() APDU with the tuner_use_status = 0x5 (inactive) for the corresponding LTSID. A udcp_status_update() APDU with the tuner_use_status = 0x5 (inactive) effectively terminates the association between a prior (active) resolve_tuning_req() and its corresponding LTSID.

In the udcp_status function, the UDCP SHALL report tuner_use_status = 0x5 (tuner inactive) for any LTSID that is not associated with a prior resolve_tuning_req().

In the udcp_status function, the UDCP SHALL only report the tuner_use_status values associated with HDD Recording (0x1, 0x3 or 0x4) when a user-initiated, scheduled, or timed recording is in effect that will always be followed by an updated udcp_status message that reports 0x0, 0x2, or 0x5, 0x6 at the end of the recording.

In the udcp_status function, the UDCP SHALL report tuner_use_status values associated with opportunistic or speculative recording that was not user-initiated as 0x6.

In the udcp_status function, the UDCP SHALL report tuner_use_status values associated with a non-displayed time-shift-buffer as 0x6.

In the udcp_status function, the UDCP SHALL report tuner_use_status values associated with live-off-disk on-screen viewing or viewing from the time-shift-buffer on DVR products as 0x0 or 0x2.

The UDCP may report "tuner_status = 0x1 tuner inactive" for tuners that are in use for non-Cable signals, such as ATSC channels.

8.6.6 tr_diagnostics

Table 33 - tr_diagnostics

Syntax	No. of Bits	Mnemonic
tr_diagnostics() {		
function_length	16	uimsbf
tr_diagnostics_revision	8	uimsbf
version_number	8	uimsbf
number_of_mmi_bytes	16	uimsbf
for (i=0; i<number_of_mmi_bytes; i++){		
mmi_byte	8	uimsbf
}		
}		

function_length Number of bytes following in this function.

tr_diagnostics_revision Version of the table format, currently set to 0x01. **Note:** All subsequent revisions are backwards compatible in that any additional parameters are added at the end of this profile.

version_number A modulo 256 counter that increments if the status in this APDU is different from the last status transmitted.

number_of_mmi_bytes Number of TR Diagnostic MMI bytes following, not to exceed 2,048.

mmi_byte TR Diagnostic MMI byte.

The TR SHALL format the Diagnostic MMI data as defined in Annex A of [CCIF2.0].

The TR MAY include multiple hyperlinks in the TR Diagnostic MMI data.

The UDCP SHALL support multiple hyperlinks in the TR Diagnostic MMI data.

9 UDCP AUTHENTICATION (NORMATIVE)

The TR SHALL perform authentication on the UDCP. This is accomplished utilizing the same Host Manufacturer certificate that the UDCP utilizes with the CableCARD.

The UDCP SHALL NOT authenticate the TR.

The authentication procedure is as follows:

The TR sends a challenge_req() APDU to the UDCP.

In response to the challenge_req() APDU, if the UDCP is currently bound to a CableCARD, the UDCP may send the existing DH_pubKey_H, $SIGN_H$, Host_DevCert and Host_ManCert to the TR in the challenge_rsp() APDU. If the UDCP is currently bound to a CableCARD or not, the UDCP may compute a new DH_pubKey_H as described in Step 1 of Figure 5.3-1 of [CCCP], or the UDCP may create a 1024 bit random number according to the requirements in section 7.2 of [CCCP] on random number generation and use this 1024 bit random number unique to the TR interface in place of the DH_pubKey_H.

The nonce, the signature, and the UDCP's X.509 certificates are sent to the TR using the challenge_req() and challenge_rsp() APDUs.

The TR verifies that the UDCP is valid by checking the nonce's signature using the RSA public key contained in the UDCP's X.509 certificates.

The TR also validates the X.509 certificates by verifying the signature of the Host Device Certificate with the Host Manufacturer Certificate Public Key, and verifying the signature of the Host_Manufacturer Certificate with the CableLabs Root Certificate Public Key. This is referred to as "chaining" to the root certificate.

If the TR validation of the UDCP certificates fails, then the TR SHALL NOT send an HMAC key.

If the TR validation of the UDCP certificates fails, then the TR SHALL report 0x02 as the authentication status in a tr_status_update() APDU within five seconds of receiving the challenge_rsp() APDU.

The UDCP can also query the TR for authentication status with a tr_status_req() APDU, as the authentication state is one of the variables in the tr_status_rsp() APDU.

The TR sends a tr_status_update() APDU after authentication succeeds, since that would represent a change in status of the TR.

10 UDCP STATUS MESSAGES (INFORMATIVE)

10.1 Channel Not Available

In the event that the UDCP requests tuning resolved for a channel that is in the channel table, but the resolve_tuning_rsp() APDU tune_status is ≠ 0x00, the message shown in Figure 3 is used as an example.

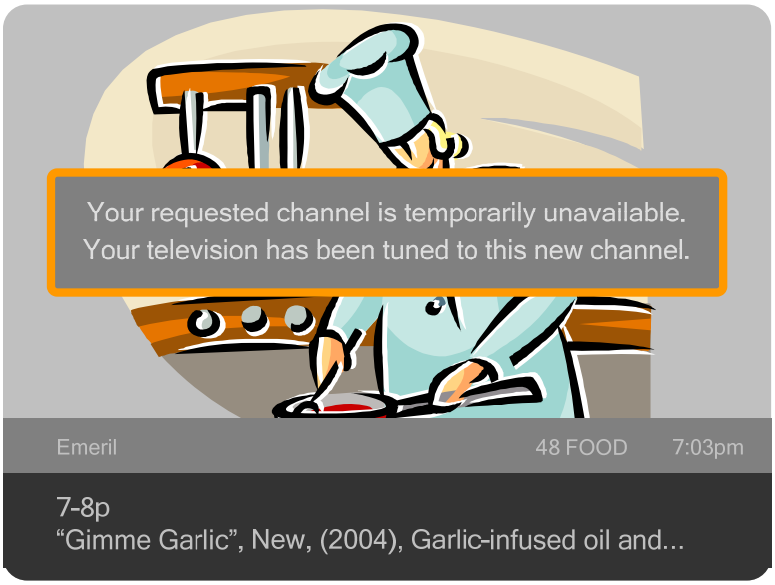

Figure 3 - Channel Not Available Message

10.2 User Inactivity Message

When a tr_message() APDU is received with the message_type = 0x00, an on-screen message like one of the following examples in Figure 4 is used. It is recommended that the user inactivity message should be displayed as a minimally intrusive overlay of active video, such as at the bottom of the screen, etc., since this message will typically be displayed during viewing of live programming. In the case where no user input is received, the display of this message may be removed by the UDCP.

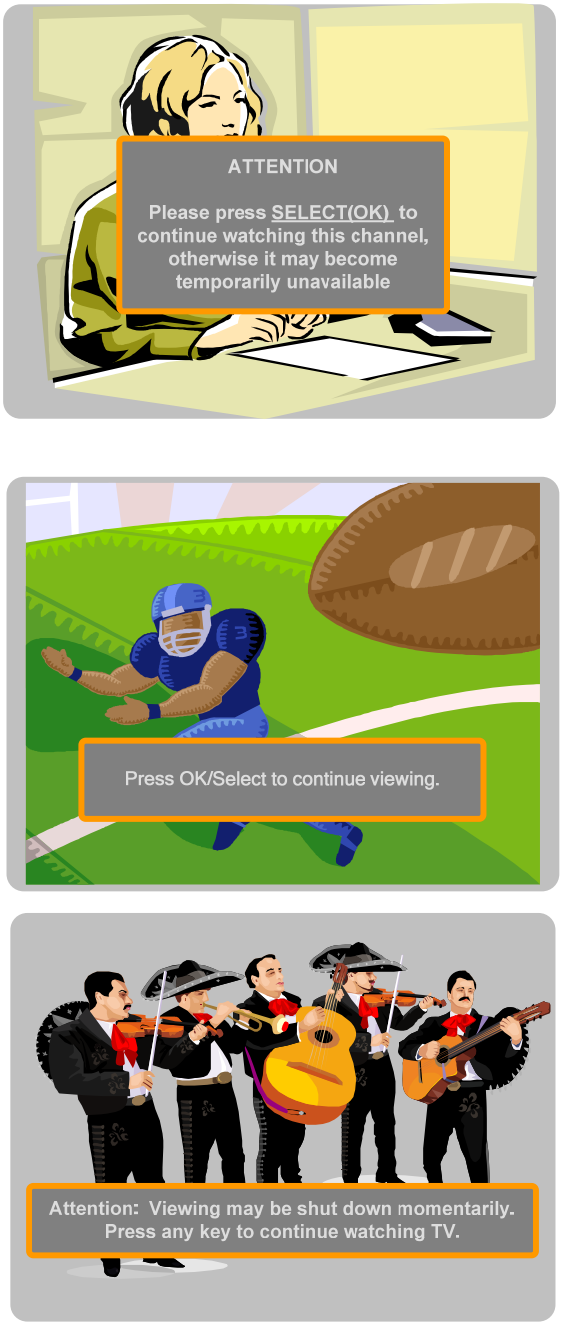

Figure 4 - User Inactivity Message Examples

10.3 Channel Specific Message

When a tr_message APDU is received with the message_type = 0x01, an on-screen message like the following examples in Figure 5 is used.

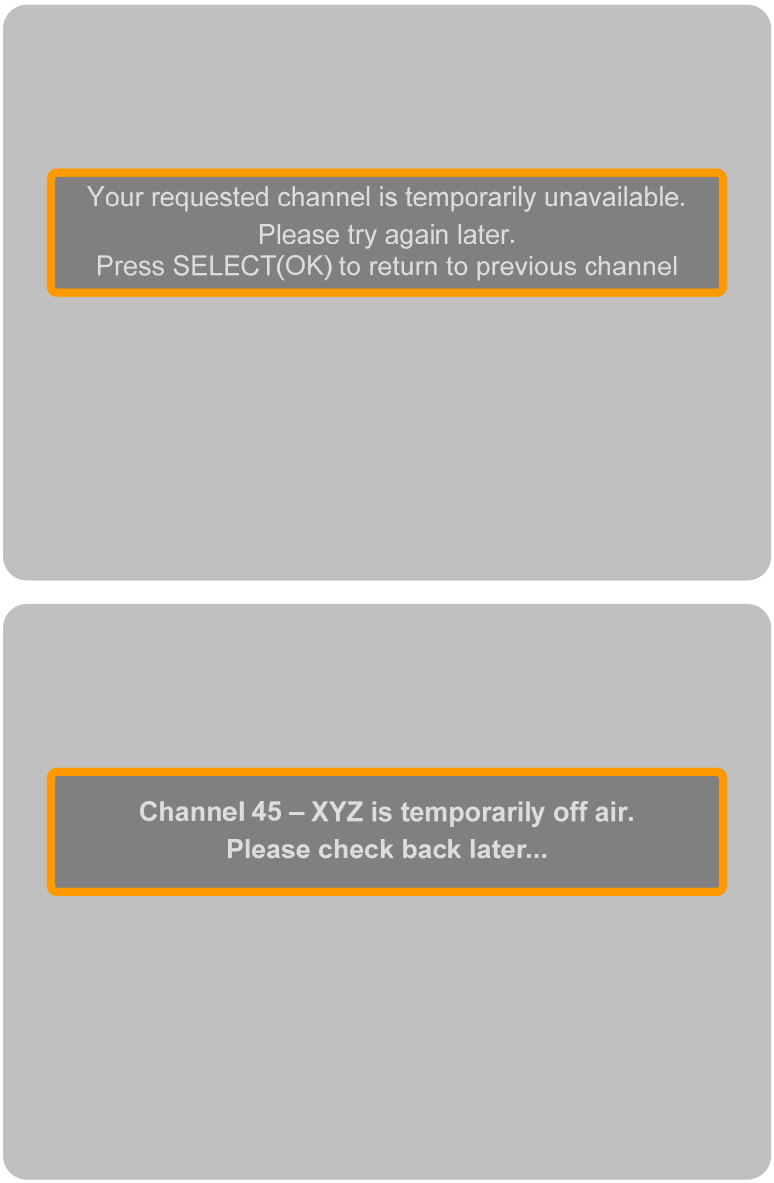

Figure 5 - Channel Specific Message Example

10.4 System Message

When a tr_message APDU is received with the message_type = 0x02, an on-screen message like one of the following examples in Figure 6 is used.

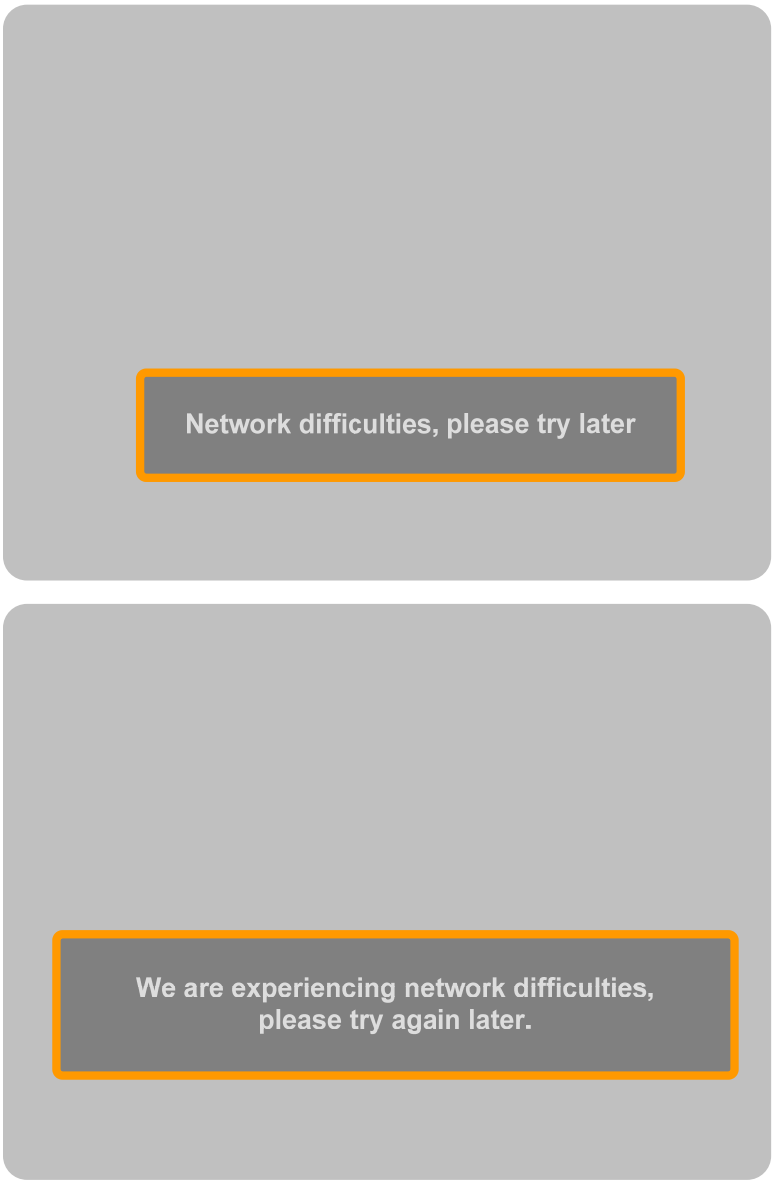

Figure 6 - System Message Example

Appendix I Use Cases

I.1 Initialization and Discovery Process

After a UDCP is powered on and performs its internal initialization to include USB enumeration, it will begin an initialization and discovery process with the TR over USB. The process will consist of three specific steps:

1. Interface initialization
   In this exchange, the UDCP and TR exchange messages to initiate communications and to each discover the capabilities of the other. If the TR does not respond within a defined period, the UDCP will retry a set number of times before declaring the TR inoperable.
2. Challenge
   In this exchange, the TR authenticates that the UDCP is allowed use of the network. If the authentication succeeds, then the TR generates an HMAC key, encrypts it with the public key from the UDCP, and sends it to the UDCP. Any bus reset will allow a subsequent attempt. The UDCP can query the TR for authentication status with a tr_status_req() APDU at any time, even after authentication fails, as the authentication state is one of the variables in the tr_status_rsp() APDU.
3. Channel list discovery
   In this exchange, the UDCP requests and receives a listing of all available channels, which supersedes any [SCTE 65] SI data.
4. Status discovery
   The TR needs to know the state of the UDCP's tuners in support of its role as a proxy for SDV state and communications. As soon as the UDCP has completed processing the channel_table_rsp, it will send a udcp_status_update message to the TR.

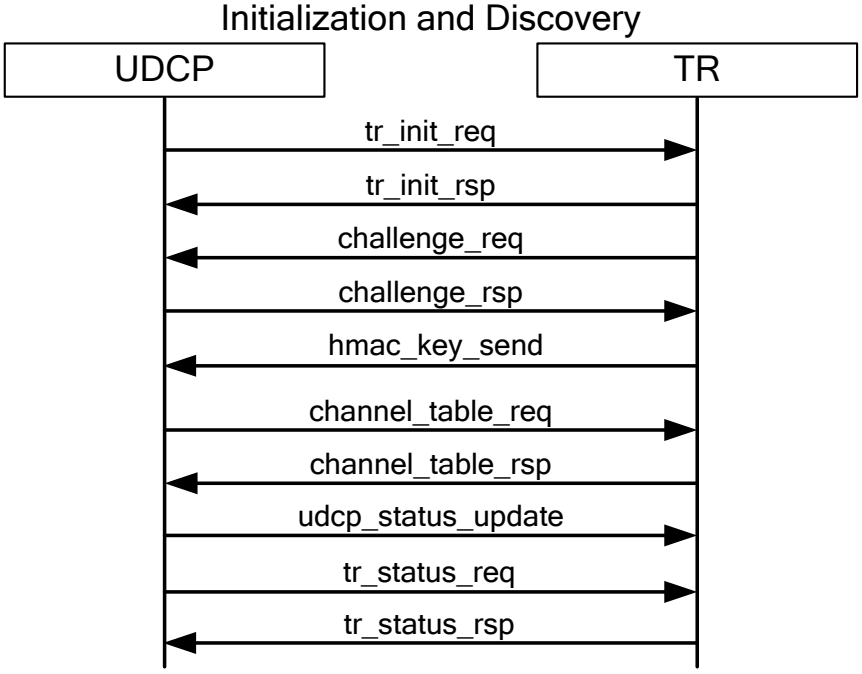

Figure 7 - Initialization and Discovery

I.2 Change to pre-existing channel listing

This use case starts when the system operator changes the available channel lineup. The UDCP needs to be informed of the change by the TR. The update to the UDCP will happen as the result of an unsolicited channel update message from the TR and without having to perform the Initialization and Discovery process.

Notes:

- The UDCP should discard the channel lineup it currently holds only after the complete new version is received and verified for integrity.
- If a new version of the channel lineup is received by the UDCP before the previous version is received in its entirety, then the previous version should be discarded.

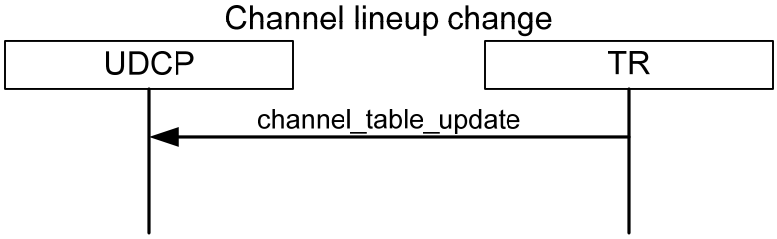

Figure 8 - Channel Lineup Change

I.3 Channel change

This use case is initiated whenever the UDCP wishes to tune to a channel defined in the channel list. Each and every channel change requires a message to the TR to resolve the tuning parameters currently associated with the channel. In either tune success or tune failure cases, the UDCP will send a resolve_tuning_cnf to include udcp_status information.

Note: User interaction or other activity can cause a status message to be sent while in the middle of tuning. The status will indicate tune in progress for the affected tuner.

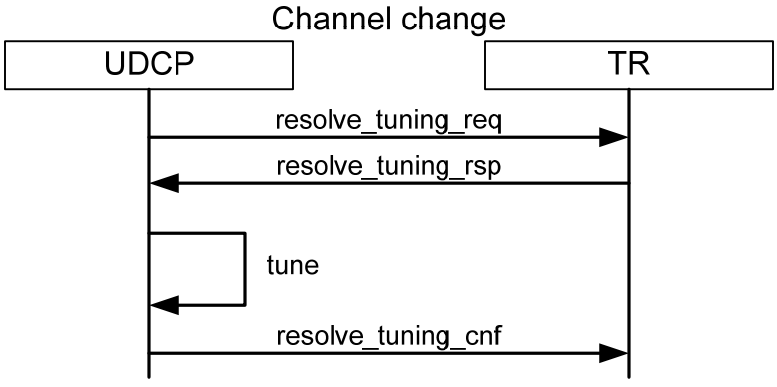

Figure 9 - Channel Change

I.4 Channel change update

Tuning parameters for an available channel may change or become invalid while a UDCP box is currently tuned. In this case, the TR will provide the UDCP new tuning parameters, and the UDCP re-tunes using the newly provided parameters.

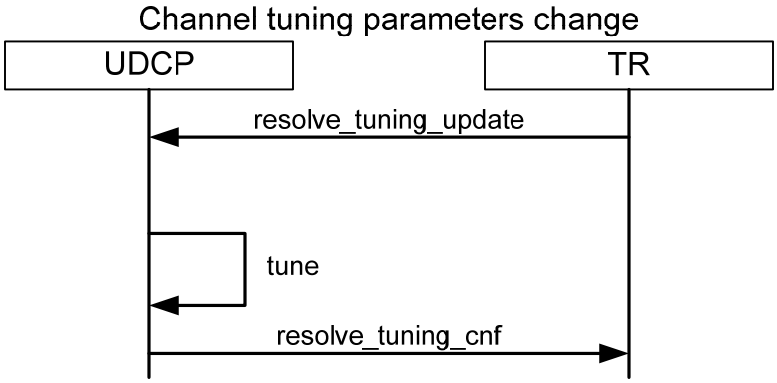

Figure 10 - Channel Tuning Parameters Change

I.5 Tuner usage change

Whenever the usage of a tuner changes, the UDCP sends a udcp_status_update message to the TR to indicate the new status.

Examples of when a status message will be sent.

- When, as a result of user interaction, a previously "live" i.e., viewed channel will only be recorded to storage media on the UDCP.
- When, as a result of user interaction, a tuner use is changed to support PIP viewing.
- When, as a result of any set of circumstances, the tuner becomes inactive.

I.6 TR Status

At any time after the Initialization and Discovery process, the UDCP can ascertain the status of the TR using the status request message flow. The TR's response will indicate the current state and viability of both upstream and downstream communications path.

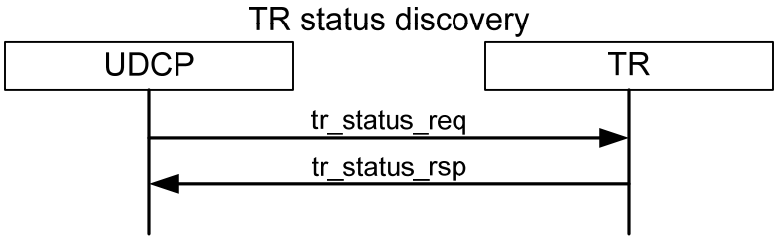

Figure 11 - TR Status Discovery

I.7 TR Status Change

At any time after the Initialization and Discovery process, the TR can send a status update to the UDCP unilaterally.

After the tr_init_req() APDU has been sent to the TR, the TR will be in the TR initializing state. It will stay in this state until it has successfully gathered the initial challenge_rsp from the UDCP, sent an HMAC key to the UDCP, and completed its initialization.

Once the TR has finished initializing, it will transition to one of the following states:

- TR ready to resolve
- TR disabled
- TR upgrade in process
- TR unavailable

Once the TR has left the TR initializing state, it will not re-enter that state until the next time it is reset, or receives a tr_init_req APDU.

The TR will enter the "TR ready to resolve" state only when it is ready to resolve tune requests from the UDCP. Before it is ready to resolve requests, the TR may stay in the TR initializing state indefinitely.

The TR will enter the "TR disabled" state if it has been disabled by the headend for any reason.

The TR will enter the "TR upgrade in process" state while it is performing a firmware upgrade.

The TR will enter the "TR unavailable" state if it becomes unavailable to resolve for a reason other than firmware upgrade, or becomes disabled by the headend.

The state diagram is:

Figure 12 - Various states of the TR as reported in the tr_status and tr_status_update APDUs.

I.8 UDCP Status

At any time after the Initialization and Discovery process, the TR can ascertain the status of the UDCP using the status request message flow. The UDCP will provide the TR a response indicating status of each tuner in use.

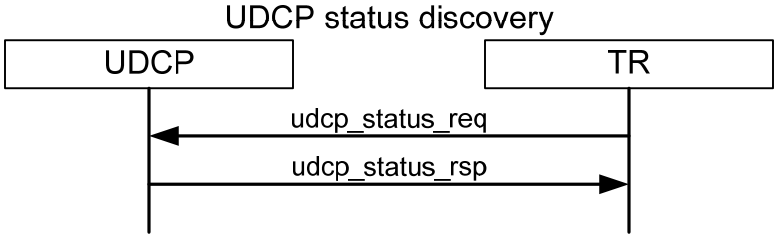

Figure 13 - UDCP Status Discovery

I.9 UDCP Status Change

At any time after the Initialization and Discovery process, the UDCP can send a status update to the TR unilaterally. The update can be sent as a result of any change in status of tuners.

I.10 TR Diagnostics

At any time after the Initialization and Discovery process, the UDCP can ask the TR for diagnostic information to aid in the troubleshooting of issues. The TR's response will be composed of MMI bytes, as defined in [CCIF2.0].

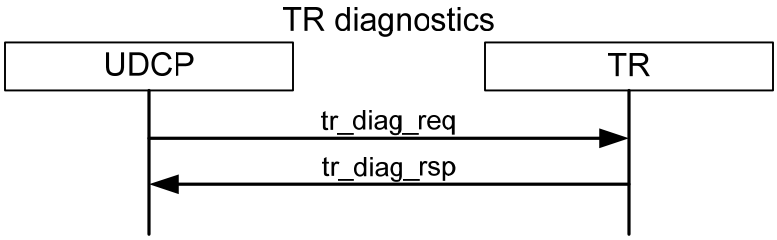

Figure 14 - TR Diagnostics

I.11 On Screen Messaging

Conditions can occur that require the system to place text on the screen of the television to which the UDCP is providing service. In this case the TR will send a message to the UDCP detailing the type of message and the text of the message desired. The UDCP can choose not to display the message, but, if so, needs to display an equivalent message. The UDCP can also respond that it is unable to display the message for reasons enumerated in the interface specification.

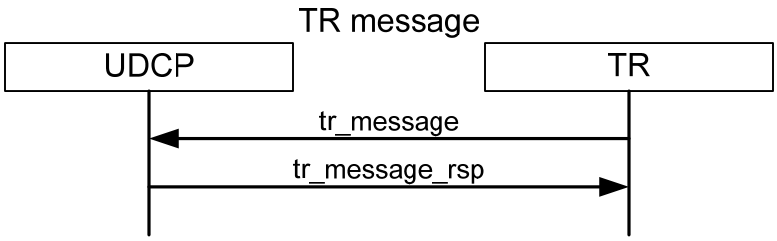

Figure 15 - TR Message

I.12 Force Tune

Conditions in the plant can arise, such as loss of a video feed, device failure, or system re-configuration that would require that the UDCP be directed to use different tuning parameters. The UDCP can be given new tuning information for the channel currently tuned or be required to tune to a different safe channel in the event that the original channel is no longer available. In this eventuality, the TR will provide the UDCP with new tuning information as per use case I.4, Channel change update.

I.13 EAS (Emergency Alert System)

The UDCP will respond to EAS requests as follows.

- If provided tuning information, the UDCP will use them to tune to the EAS channel.
- If provided a source ID, the UDCP resolves tuning parameters by interacting with the TR.

I.14 TR Firmware upgrade

The TR will notify the UDCP that it will be "off-line for a while" when performing a TR firmware upgrade and signal when it has finished and is once again ready via tr_status_update() APDUs, similar to how CCIF homing resource APDUs are used by the CableCARD to notify its host during a CableCARD firmware upgrade.

I.15 Independent start-up of both TR and UDCP

Both devices support the independent start-up and shut-down of the other device without an adverse user experience. Both devices will support an unscheduled disconnection of the USB interface without an adverse user experience.

I.16 Start-up Use Cases

The expected standard use case for start-up is that both the UDCP and TR are powered on and initialized independently before the TR is plugged into the UDCP via the USB cable. This standard use case should be recommended to users.

Both devices need to be able to support the following use cases as well:

- The un-powered TR is plugged into the active UDCP and then the TR is powered on.
- The powered TR is plugged into the un-powered UDCP and then the UDCP is powered on.
- The un-powered TR and UDCP are connected via USB and then both powered on.

In all start-up use cases the TR initializes the USB communication channel after the TR has completed its power-up and cable network initialization sequence and the UDCP is detected on the USB bus. If the UDCP USB interface is not detected or not active, the TR will wait until the UDCP USB interface is active. Proper setup of the TR solution can require new entitlements to be sent to the CableCARD in the UDCP. The installer or consumer may need to contact the cable operator to obtain the correct entitlements.

I.17 Independent shut down of both TR and UDCP

The expected standard use case for shut-down is that the USB cable is disconnected and the TR is then powered down by the user, while the UDCP remains powered and active. The UDCP discovers that the TR is no longer connected and adjusts to the lack of a tuning resource. The UDCP can present information to the user indicating that the TR is not available and SDV services are no longer available.

Both devices need to be able to support the following use cases as well:

- The UDCP is powered down or placed into standby while the TR remains powered and active. In this case the TR waits for the UDCP to be powered up again and resumes normal behavior. Once the UDCP is detected, the USB interface will be re-initialized and resume normal behavior.
- The UDCP is powered down and the TR is disconnected. In this case, when it is powered up, the UDCP discovers the TR is no longer connected and adjusts to the lack of a tuning resource.

I.18 The UDCP Channel Map

The UDCP will not use any channel map previously provided by the TR when:

- the TR has been disconnected from the UDCP
- the TR has indicated that it is temporarily unavailable because of a software update or other reason
- the TR fails to certify the UDCP

The UDCP can use a channel map from the CableCARD or other source in these cases. The UDCP can indicate to the user that the TR is not available and SDV services are no longer available. In the case that the TR fails to certify the UDCP, the UDCP user interface can recommend the user contact their cable operator.

Appendix II Revision History

The following ECNs were incorporated into version I02 of this specification:

ECN	Date Accepted	Title
TRIF-N-08.1190-1	3/14/08	UDCP Authentication Failed Requirement
TRIF-N-08.1201-3	4/11/08	Clarify when a new HMAC key can be sent
TRIF-N-08.1207-1	4/11/08	Remove incorrect language in the appendix about communication being halted
TRIF-N-08.1209-1	4/11/08	Clarify the number of possible USB vendor and product ID combinations
TRIF-N-08.1213-1	4/11/08	Fix several miscellaneous editorial errors
TRIF-N-08.1215-1	4/11/08	Clarify that TR uses TRIF APDUs and not CCIF APDUs for upgrade notification
TRIF-N-08.1220-2	4/11/08	UDCP Response to new HMAC Key
TRIF-N-08.1225-2	4/11/08	Relax size restriction on tr_diagnostics MMI data
TRIF-N-08.1235-1	4/11/08	Resolve ambiguities regarding how UDCP notifies TR that it has tuned away
TRIF-N-08.1236-4	4/11/08	Clarify when the TR is ready to respond to tuning requests
TRIF-N-08.1203-2	4/18/08	Remove redundant message requests from the TR
TRIF-N-08.1205-2	4/18/08	Clarify the requirement that the TR always support at least two tuners
TRIF-N-08.1206-3	4/18/08	The UDCP doesn't need to report user activity if the tuners are inactive
TRIF-N-08.1211-2	4/18/08	Add machine readable TR diagnostic information to aid troubleshooting in the field
TRIF-N-08.1217-4	4/18/08	Clarify when the UDCP displays a diagnostic message from the TR
TRIF-N-08.1240-1	4/18/08	Clarify that TR shall reject LTSIDs from UDCP that exceed number of tuners supported by UDCP

The following ECN was incorporated into version I03 of this specification:

ECN	Date Accepted	Title
TRIF-N-08.1329-1	1/16/09	Add requirement for UDCP to support multiple hyperlinks in TR Diagnostic MMI data