RDD 10-2006

SMPTE REGISTERED DISCLOSURE DOCUMENT

An Open Transport and Navigational Specification, Optionally Supporting Multiple Conditional Access Systems

The attached document is a Registered Disclosure Document prepared by the proponent identified below. It has been examined by the appropriate SMPTE Technology Committee and is believed to contain adequate information to satisfy the objectives defined in the Scope, and to be technically consistent.

This document is NOT a Standard, Recommended Practice or Engineering Guideline, and does NOT imply a finding or representation of the Society.

Errors in this document should be reported to the proponent identified below, with a copy to eng@smpte.org, p.dare@worldnet.att.net.

All other inquiries in respect of this document, including inquiries as to intellectual property requirements that may be attached to use of the disclosed technology, should be addressed to the proponent identified below.

Proponent contact information:

Lee Pedlow
Sony Electronics INC
16530 Via Esprillo,
San Diego,
CA. 92127

E-mail: lee.pedlow@am.sony.com

Approved
September 6, 2006

Introduction

This specification is in two parts: Part 1 deals with the transportation and management syntax of a tool kit that providing basic access control and optionally allowing for multiple Conditional Access (CA) systems to coexist; Part 2 of the specification deals with the navigational tools.

This technology was developed to provide basic management of devices accessing basic-tier program content, optionally coexisting in the presence of one or more conditional access systems. Unlike other attempts to address this issue, the basic access control and one or more CA systems operate within a common MPEG-2 transport stream without any direct or indirect interaction between the CA systems.

Much of the syntax and data structure is based upon the ETSI-DVB model and related standards.

Applications of this specification are potentially any point to multi-point content distribution requirement such as network program distribution, theater content distribution, as well as applications in the cable and Satellite industries.

Part 1 – Transport and Basic Access Control System Specification

Table of Contents

List of Figures – Part 1

1 Scope

The intent of this specification is to provide detailed system-level specifications for the basic access control subsystem and algorithms for application in both the source equipment and EUE devices. This technology is applicable wherever an MPEG-2 transport stream is used for content delivery.

1.1 Overview

The purpose of the initiative, promoted by Sony, is to facilitate competition in an open marketplace by providing a means to deploy non-legacy source equipment, subscriber devices and services on existing legacy networks or distribution channels. The navigation data specification forms part of a tool kit allowing content owners and network operators to implement a method for digitally distributing content that was previously analog and to optionally choose alternative CA vendors while still preserving legacy choices. Migration paths to up-grades are also possible. Having the capability to support the coexistence of multiple CA systems without the need for content replication can be a bandwidth saver in many installations.

1.2 All-Digital Content Carriage

The enabled all-digital system implements a standards-based method to provide basic access control, to implement remote authorization and management of the EUE/receivers. This achieves the objective for centralized connection management, something that is common with network distribution and in the cable industry. The basic access control system can operate in conjunction with traditional 3rd party CA systems, thereby enabling open-market alternative devices to be employed.

1.3 Reference Documents

Unless a specific version or revision number is included, the following references are non-specific, and the latest version applies.

[1] ISO/IEC 13818-1, Information Technology — Generic Coding of Moving Pictures and Associated Audio Information: Systems — Part 1: Systems, ISO, 4/02

[2] ETSI EN 300 468 v1.4.1 (2000-11), Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB Systems

[3] ETSI ETR 289 ed.1 (1996-10), Digital Video Broadcasting (DVB); Support for Use of Scrambling and Conditional Access (CA) within Digital Broadcasting Systems

[4] Specification of the Macrovision Copy Protection Process for EUE/IRD Products, v7.1.S1, 1999, Macrovision Corporation

[5] ETSI TS 103 197, Head-end Implementation of DVB Simulcrypt, v1.3.1 (2003-01)

[6] ETSI EN 301 192, DVB Specification for Data Broadcasting, v1.3.1 (2003-05)

[7] ANSI J-STD-042-2002, Emergency Alert Message for Cable, American National Standards Institute, 12/3/02

[8] FIPS PUB 180-2, *Secure Hash Standard*, Information Technology Laboratory, National Institute of Standards and Technology, August 1, 2002

[9] FIPS PUB 198, *The Keyed-Hash Message Authentication Code (HMAC)*, Information Technology Laboratory, National Institute of Standards and Technology, March 6, 2002

[10] FIPS PUB 6-4, *Counties and Equivalent Entities of the United States, Its Possessions and Associated Areas*, National Institute of Standards and Technology, January 24, 2002

[11] Fowler/Noll/Vo (FNV) Hash, www.isthe.com/chongo/tech/comp/fnv

[12] EUI-48™,http://standards.ieee.org/regauth/index.html

1.4 Acronyms and Abbreviations

ANI	Automated Number Identification
AV&V	Automated Validation & Verification
BAC	Basic Access Control
Bslbf	bit serial, leftmost bit first
CA	Conditional Access
CAS	Conditional Access System
CASID	Conditional Access System Identifier
CAT	Conditional Access Table
CMTS	Cable Modem Termination System
DDC	Download Data Carousel
ECM	Entitlement Control Message
EMM	Entitlement Management Message
EPG	Electronic Program Guide
EUI	Extended Unique Identification
EUE	End-User Equipment (STB, IRD's, Professional decoders)
FIPS	Federal Information Processing Standards
HMAC	Hash Message Authentication Code
IC	Integrated Circuit
MPEG	Moving Pictures Experts Group
MSO	Multiple System Operator
NTS	Network ID, Transport ID and Service ID
OSD	On-Screen Display
OUI	Organizationally Unique Identification
PAT	Program Allocation Table
PID	Packet Identifier
PMT	Program Map Table
PPV	Pay-per-View
PSI	Program Specific Information
PUSI	Payload Unit Start Indicator
QAM	Quadrature Amplitude Modulation
SBS	Slow/Simple but sure
SHA	Secure Hash Algorithm
uimsbf	unsigned integer, most significant bit first

2 Content Access Control

2.1 Basic Access Control Concept Overview

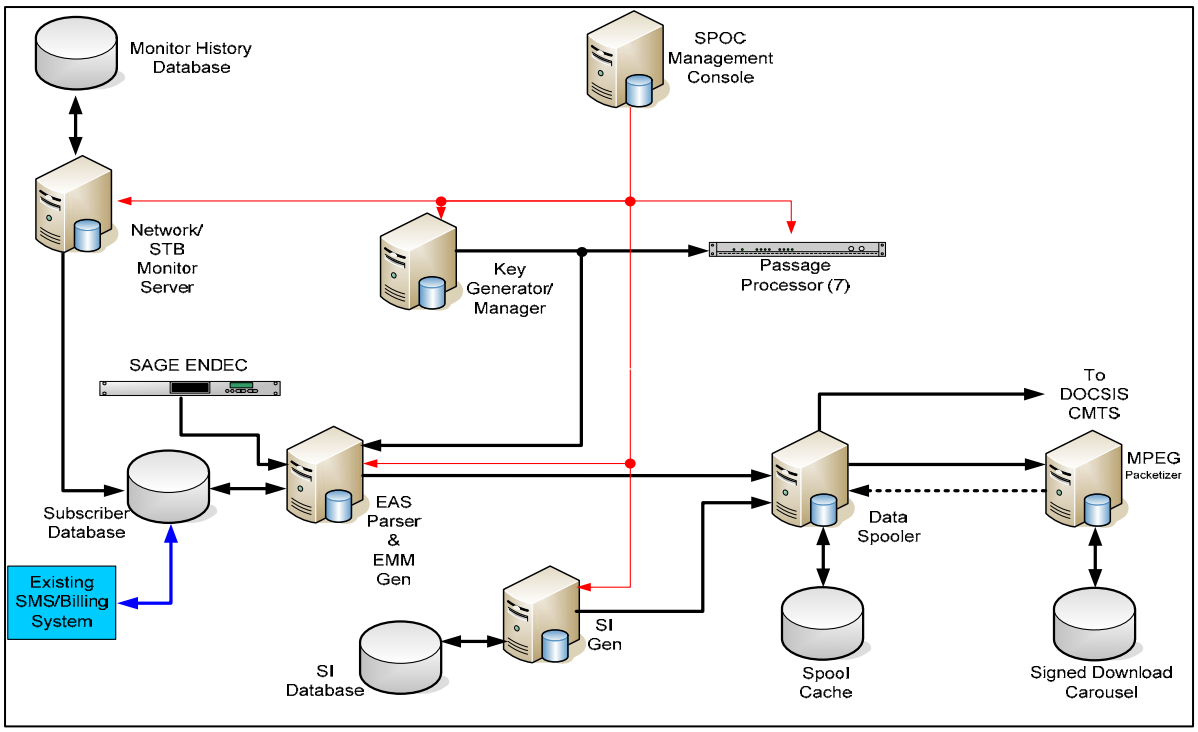


Figure 1 – Typical Cable BAC System Architecture

The purpose of the basic access control system (BAC) is to provide the ability to remotely manage connections to distribution networks. Under the basic access control premise, all digital content in the basic service tiers can be processed using a technique such as Passage® to support legacy devices in conjunction with the legacy conditional access system. Open market devices implementing the tools provided in this specification can then access the basic-tier content, under the central management of the basic access control system, without the requirement to implement a specific CAS are also encrypted using the widely accepted and deployed ETSI-DVB-CSA standard.

The basic access control system provides for dynamic replacement of the content keys used to encryptecrypt, with a key epoch that is significantly longer than that used in a traditional CA system but without the complexity, costs and additional messaging systems associated with those systems. Key delivery and EUE authorization is performed through the delivery of an entitlement management message (EMM), delivered through both cable DOCSIS, for two-way devices, and in-band, supporting one-way devices.

The EMM format supports both unit (singular) and group messages and both the definition of and membership within a group can be arbitrarily defined. The devices under BAC management are able to belong to multiple groups and each device by default belongs to two groups: itself (unit) and a global group encompassing all devices. The EMM is structured as a standard MPEG-2 private section, containing one or more descriptors. The descriptors provide specific direction to the EUE regarding channel authorization, key delivery, software download commands, group membership, emergency alerts, etc. Regardless of number or type of descriptors contained within the EMM, each message is both encrypted using a global, fixed key and validated through a cryptographic signature.

The BAC system provides the following EUE management features:

- Support of both one-way and two-way devices
- Addressable
    - Unit Addressable
    - Group Addressable (up to 255 discrete groups assignable per device)
- Renewable content keys, with a “key per service”, wherein the same key is used for both the audio and video components of a given service.
- Scheduled, boot-time or immediate software download
- Multiple service tier support (up to 65535)
- “Opt-out” á la carte service masks
- Selectable Macrovision anti-taping enablement per service, per device masks
- SCTE/ANSI EAS support with local trigger for cable applications
- Individual FIPS code provisioning

All message authentication is done in software. Every cryptographic system must have a root secret stored somewhere inside the EUE client. In traditional CA systems, this is done is either smart cards, dedicated custom ICs or through other means to create an electrically and mechanically secure environment to store the root secrets.

The basic access control system does not use these extremely complex schemes nor incurs the associated hardware and licensing costs for managing access to basic tier content. The system contains four root secrets that are stored within the EUE. In order to compromise the system, all four must be known. The root secrets are the respective key pairs for the EMM and SW download transport encryption and the keys for both the SW download and EMM HMAC algorithms, respectively. Though outside the scope of the present activity, the BAC system could be easily extended to utilize unique, proprietary features contained within a particular MPEG-2 decoder IC design for additional security and robustness. This would be accomplished by allowing storage of the root secrets only in an encrypted form within the EUE and only allowing them to exist in their unencrypted state within an electronically and mechanically secure area within the decoder IC silicon using the features referenced above, should a business case requiring such be identified.

2.2 Architecture

The basic access control schema shall use at least one key pair per service tier. Service tiers are groups of individual logical channels associated with services that are bundled as a discrete entity (“package”). Though outside the scope of this specification, this methodology is extensible to support service tiers of any size, including single programs, such as individual events. Access to a particular service tier is determined by whether a supported EUE device has possession of the appropriate service tier key(s). To maintain message security, all EMMs shall be signed by a private key so that the EUE device may authenticate the received message.

In addition, entitlement (authorization) to individual services is determined through a ***service_mask***, matched to each service with an authorization bit carried in the mask in an “opt-out” fashion. If a particular service bit is one, the content represented by the bit is non-authorized. The ***service_mask*** is maintained in nonvolatile memory within each supported EUE device and is locally modified through data carried in authenticated EMMs to reflect the services that a particular EUE device is intended to access. This arrangement, while reinforcing the key-per-tier schema, also allows implementation of á la carte programming, should it be required to pursue a particular business model. This schema also allows for the selective removal of programming from a particular EUE upon subscriber request.

2.3 Encryption Keys

Three types of keys shall be defined and employed: **emm_key**, **download_key** and **content_key**.

2.3.1 EMM Keys

The **emm_key** is a pair of 64-bit values that shall be fixed for the four elements of the *Simultrans* phase-II activity defined in this document. It shall be used to decrypt all ETSI-DVB-CSA encrypted EMM messages received by the and shall be the same value for all devices. These values shall be permanently stored in the EUE in a secure manner. The values shall be defined by the application.

2.3.2 Download Keys

The **download_key** is a pair of 64-bit values that shall be fixed for the four elements as defined in this specification. It shall be used to decrypt all ETSI-DVB-CSA encrypted SBS-download content messages received by the EUE and shall be the same value for all devices. These values shall be permanently stored in the EUE in a secure manner. The values shall be defined by the application.

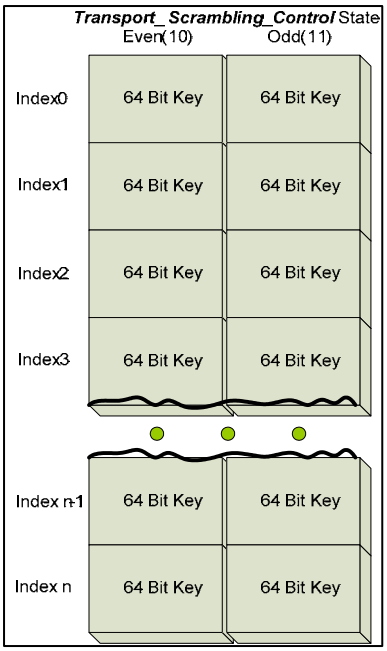


Figure 2 – Content Key Structure

2.3.3 Content Keys

Keys for encrypted content shall be stored in the EUE in nonvolatile memory in a secure manner. The content keys are organized as an array, as shown in Figure 2. A suggested structure for the array in the EUE is as follows:

```
UINT16 key_index;
BOOLEAN  key_phase;
UINT64  content_key[key_index, 2];
if (transport_scrambling_control == '10')
    key_phase = 0;
else
    if ((transport_scrambling_control == '11')
        key_phase = 1;
```

The maximum possible storage requirement for this structure is 1 MByte, though it is highly unlikely that a system would have 65,536 unique keys. The same content key shall be used for both audio, metadata and video elements of a particular service. The association of a particular key pair (even and odd phase) to a specific service is done through data contained in the table delivered through the program authorization descriptor, mapping a logical channel to a ***key_index*** value, which is then used to find the corresponding content key pair in the content key array. This relationship is illustrated in Figure 3. The table containing the list of logical channels and the associated key indices shall also be securely stored in nonvolatile memory within the EUE.

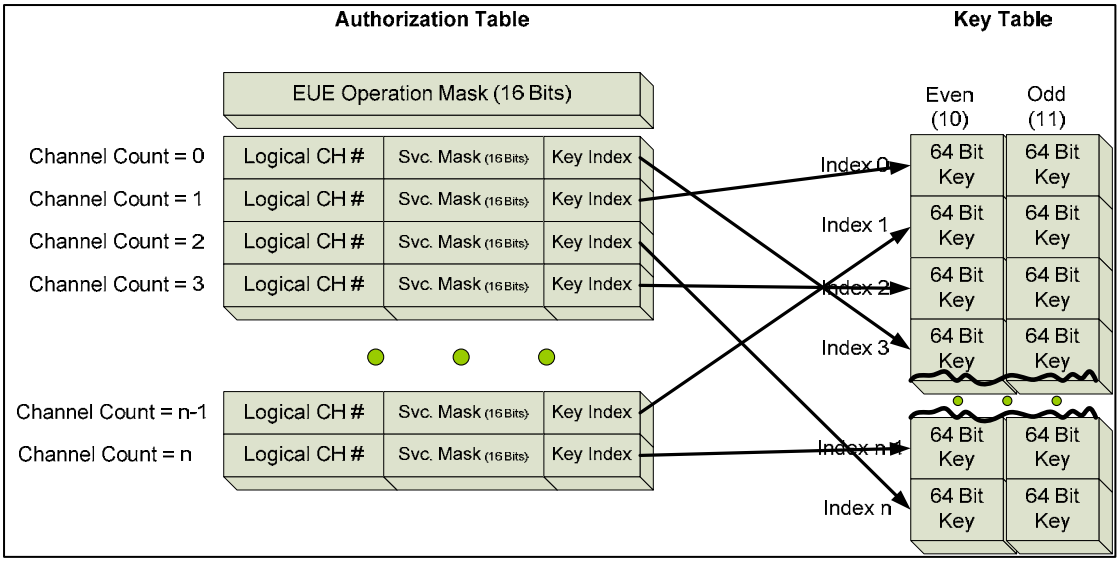


Figure 3 – Association of Content Key to a Service

The content keys may be delivered dynamically. The content key array structure shall be implemented as described above and could optionally be pre-loaded with static data as part of the software image downloaded to the device. For these elements, the content key array may be physically located in a different memory resource or a different portion of a memory resource.

Likewise, the data structure containing the ***logical_channel***, ***key_index*** and ***service_mask*** values shall be a hardcoded table organized in the same manner as if a channel authorization descriptor had actually been received in an EMM. This data shall be embedded in the downloaded software image. The same caveats as indicated for the content key structure apply regarding storage of this structure.

2.3.4 Content Key Delivery

Content key delivery is performed through the receipt of an EMM containing a valid key data descriptor. Content key data is organized as a table containing the pair of content key values covering both MPEG-2 encryption phases (even and odd). These content key pairs are associated with an index identifying a

particular content key pair. The key data descriptor structure allows for the en masse replacement of the entire column of content keys in the table associated with a particular encryption phase. A single descriptor cannot replace both the content keys in a particular key pair. Two independent messages must be sent to populate or refresh the entire content key table, one identified as associated with the even encryption phase and then one identified as associated with the odd encryption phase. The order of these messages is unimportant.

Additionally, an optional third element may be employed to increase the level of indirection in transmitted data. The key index descriptor allows the key index to be transmitted separately from the program authorization table and the key data tables. This descriptor overwrites index data in the program authorization table and can be used to initially provision index data separately or to rekey a system solely through transmission of new indices without transmitting key data or program authorization tables. The EMM carrying the complete indices for a program tier would be very small and is likely contained in a single packet, being 1/3 the size of the authorization table and at a minimum 1/4 the size of the key data table.

The EUE device shall never alter or modify in any way the format or contents of the data structures containing the program authorization and the key data tables, except to apply the appropriate descriptor data from a validated EMM. The EUE device shall place the data received from the relevant EMM in the exact form and order into the storage structure and in no event sort or alter the order of entries in the table. It shall be incumbent upon the BAC system to provide the logical channel entries in the program authorization table in a sorted fashion. Each program authorization table and the key data table shall be transmitted in a complete form and shall replace any existing table in its entirety.

In practice, the key data message shall be used to update only the future content key, which would be the keys associated with the phase opposite of the one currently being used to encrypt content. This method allows background provisioning of authorized s with the content keys for the next epoch without any impact upon current device operation.

2.4 Signatures

Electronic signatures shall be employed to authenticate entitlement messages and software download messages. The signature method employed is the published SHA-1 hash **[8]**. This process produces a unique 160-bit condensed cryptographic checksum of the evaluated message or image. The hash value is then encrypted using a published algorithm (HMAC) **[**Error! Reference source not found.**]**, but employing a secret key. The combination of the one-way hash and the secret encryption prevents both alteration of the original message or image and substitution of both content and a corresponding hash. The secret 512-bit key value used for EMMs shall be different from the value used for software download.

2.5 Device Addressability

The basic access control system architecture provides a simple yet powerful method for addressing EUE devices based upon the EUI-48 format of **IEEE-802 [12]**. The scheme supports global, unit and group addressing modes. There shall be no distinction between messages addressed as either group, global or unit with respect to scope or priority. With the exception of the ***table_id*** value, all messages types are identical in structure and therefore can be handled by the same parsing, validation and evaluation processes.

2.5.1 Global Addresses

Every EUE by default shall belong to a single, universal group. This group membership is irrevocable and provides at least one method to address all EUE devices in the system. The universal group address is a constant and is EUE defined as **0x100000FFFFFF**. The EUE shall monitor for messages bearing a private table ID of 0x84 and containing the global group address. If a message matches those criteria, the message is then passed in its entirety to the host application for validation and further evaluation.

2.5.2 Unit Addresses

Every device, including one-way devices, shall have a unique EUI-48 compliant MAC address assigned to the A/V decoder. This MAC address shall be in addition to a separate MAC address associated with an internal DOCSIS compliant modem, present in two-way devices. The MAC address shall be immutably stored in nonvolatile EUE memory and shall not be alterable during reset, software download, message storage, etc. The MAC address associated with the A/V decoder shall be used as the unique identifying element of that EUE asset in the system. The decoder MAC address is the address referenced in unit addressed EMMs. Upon receiving an EMM and decrypting it using the EMM decryption key, as described earlier, the EUE shall filter for new messages bearing its unit address. If a message matches those criteria, the message is then passed in its entirety to the host application for validation and further evaluation.

2.5.3 Group Addresses

In addition to the global and unit addresses that each EUE has assigned, a may have assigned an address signifying membership in a group so that messages may be sent to predefined subsets of devices, as determined by the source/MSO. Each device may have concurrent membership in multiple groups, with each EUE supporting up to 256 possible simultaneous memberships. Association of a EUE to a particular group or multiple groups shall be done through an EMM containing a group assignment descriptor. The group assignment descriptor carries an array of group addresses, with which the addressed EUE shall be associated. The receipt of a new group assignment descriptor completely overwrites all previous group assignments, stored in EUE nonvolatile memory. Receipt of a validated group assignment descriptor with the ***group_count*** field set to zero shall signify revocation of all group memberships, except the irrevocable global address, hardcoded in the application. All group addresses shall have the OUI field value of **0x100000** and no group shall have bits 24 to 40 (following the OUI) of the ***group_address*** field assigned as **0xFFFFFF**, since this value is already assigned for the global address.

2.6 Content Access Management

2.6.1 Channel Maps

Channel maps are defined using a combination of two standard elements. Per the ETSI-DVB architecture, all RF network resources are EUE defined within the SI construct Network Information Table (NIT). The NIT describes the RF parameters associated with each service based upon 'NTS' criteria (Network ID, Transport ID and Service ID). The composite value for NTS must be unique for every service available within the system.

The ETSI-DVB structure Bouquet Association Table (BAT) provides a linkage between logical channel numbers and a specific NTS for NIT lookup and determination of the necessary RF parameters to access the QAM signal carrying a desired service. There may be multiple BATs, providing multiple channel line-ups that could represent differing channel plans based upon service tier, region, etc. Each BAT has an associated bouquet ID that uniquely identifies each BAT. All EUE shall be assigned an applicable bouquet ID upon provisioning via EMM indicating the possible authorized services that could potentially be accessed by a particular EUE.

By default, all EUE shall use a bouquet ID of zero when no bouquet ID has been issued by the headend/source. There are only two cases that this case should be encountered. The first case is when a new device is connected to the system prior to the headend/source sending (or the EUE receiving) provisioning. The second case is when an EMM is received from the headend/source with an operation mask setting indicating that the EUE should be deauthorized. The process of deauthorization is defined as deleting the bouquet ID variable along with the contents of the program authorization table.

A bouquet ID value of zero shall be reserved for channel map limited to the carriage of informational or promotional content channels such as barker channels.

2.6.2 Service Access Authorization

Service access is managed through the program authorization table, a logical construct maintained within the nonvolatile memory of each EUE device and its contents delivered via EMMs from the source. This table contains a list of logical channels that the particular device is permitted to access, along with the key index and other data uniquely associated with each service. An illustration of the structure is shown in Figure 4. The presence of a logical channel in the channel list contained within the program authorization table provides indication that the channel is a candidate for viewing on a particular EUE, but does not in itself guarantee access.

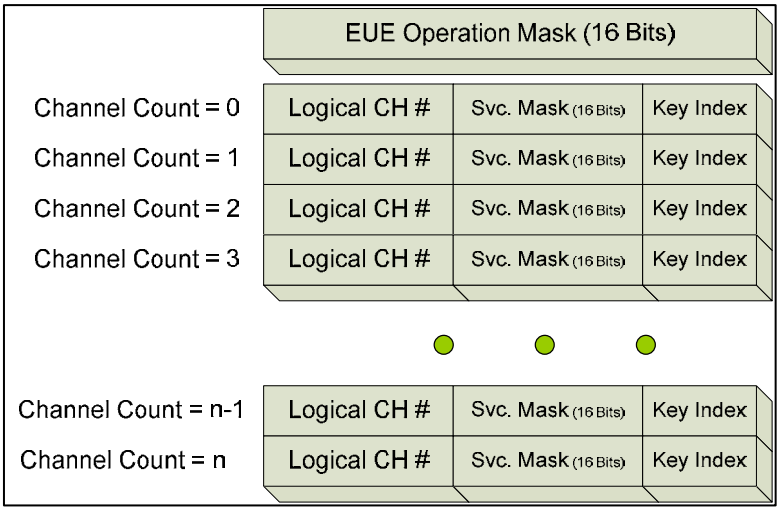


Figure 4 – Program Authorization Table Structure

The EUE device shall never alter or modify in any way the format or contents of the data structures containing the program authorization and the key data tables, except to apply the appropriate descriptor data obtained from a validated EMM. The EUE device must place the data received from the relevant EMM in the exact form and order into the storage structure and in no event sort or alter the order of entries in the table. It will be incumbent upon the BAC system to provide the logical channel entries in the program authorization table in a sorted fashion. Each program authorization table shall be transmitted via EMM to the EUE device in a complete form and shall replace any existing table contents in its entirety. No partial channel maps shall be transmitted and received table data shall never be appended within the EUE. The EMM containing the program authorization table information will not contain the associated service and operation masks. These items shall be delivered by separate EMMs and shall be composited by the EUE onto the program authorization table.

Two other components, combined with the presence of a service's logical channel number in the program authorization table, control the ultimate determination of whether the EUE should be able to access a particular service. One component is the ***channel_unauthorized*** bit contained within the ***service_mask*** variable associated with each logical channel entry in the program authorization table. If this bit is set, accomplished through an EMM, then the particular channel is deauthorized and not available for viewing. The other controlling components are the ***EUE_deauthorized*** and ***brick_mode*** bits contained within the EUE ***operation_mask***, which controls attributes for the entire EUE. It too is managed through EMM messages and is part of the program authorization table structure.

2.6.3 Service Access Process Flow

The process of accessing a particular service is shown in Figure 5 and follows the subsequent logic:

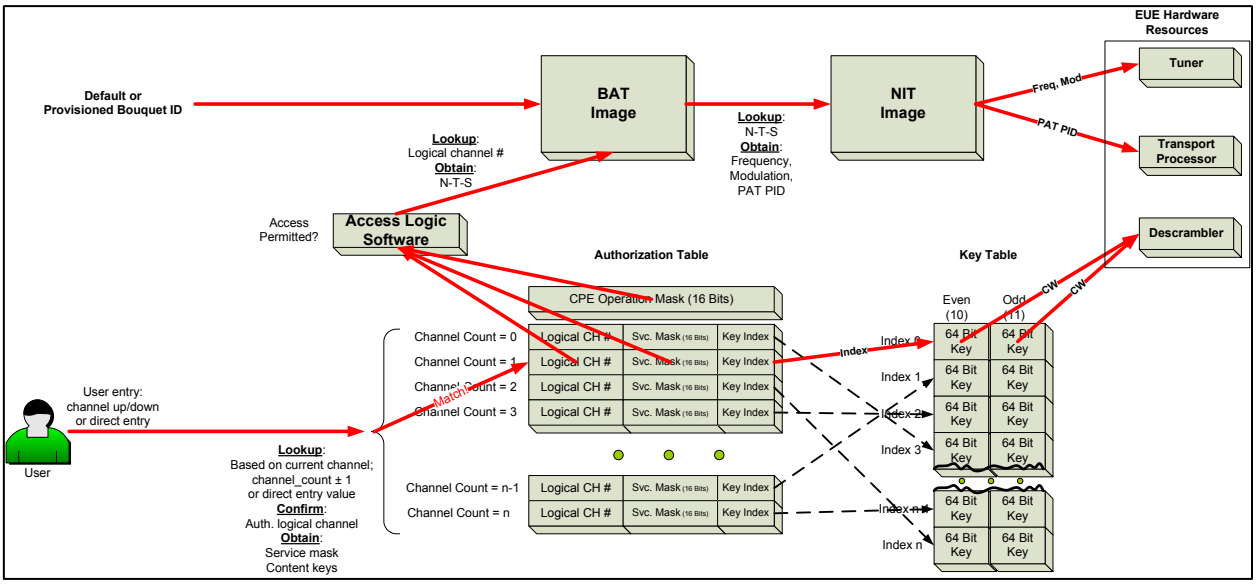


Figure 5 – Channel Access Process Flow

1. A viewer indicates the desire to change decoded A/V service through their remote control. The indication consists of either a channel upown command or a direct logical channel number entry.

2. The EUE application receives, decodes and validates the remote control message.

3. The EUE application checks the value of the ***operation_mask*** contained within the program authorization table to see if either the ***EUE_deauthorized*** or ***brick_mode*** bits are set. If the either of these bits are set, further processing of the request is ended. If the ***bouquet_id*** is set to zero (default), then the process continues without further reference to the program authorization table for access to free or barker channels in an unprovisioned EUE and goes directly to step 7.

4. The EUE application parses the ***logical_channel*** field of the program authorization table using the following index:

   a. For channel upown, use current channel number as the search index. When the match is found, increment or decrement one entry as appropriate to reach the entry for the target channel.

   b. For direct entry, use the entered logical channel value as an index for the target channel. If the desired target channel is not contained within the table, display and appropriate OSD informing viewer.

5. From the program authorization table, the EUE application obtains the service mask field associated with the logical channel entry in the program authorization table and checks to see if the ***channel_unauthorized*** bit is set. If this bit is set, then further processing of the request is ended. The application also obtains the ***key_index*** value associated with the requested service from the same record in the program authorization table containing the target ***logical_channel*** value.

6. Using the ***key_index*** value associated with the requested service, the application accesses the key table stored in nonvolatile memory, using the ***key_index*** value as the array index to obtain the two 64-BIT ETSI-DVB-CSA working keys used to encrypt the content. These values are loaded by the EUE application in the transport decryption subsystem for content decryption.

7. The EUE application parses the BAT image stored in RAM, using the ***logical_channel*** and ***bouquet_id*** values as paired indices. When the entry in the table matching both criteria is found, the associated ***network_id***, ***transport_id*** and ***service_id*** values (referred to collectively as 'NTS') are obtained from the table.

8. The EUE application next parses the NIT image stored in RAM, using the NTS values obtained from the BAT as composite indices. When the entry in the table matching all criteria is found, the associated frequency, modulation and PAT PID information are obtained from the table. The data obtained are then used by the EUE application to provision the tuner, demodulator and transport processing subsystems to access the transport containing the requested service.

9. The EUE executes the tuning process and then follows the standard MPEG procedure using the ***service_id*** of the requested service and the transport PSI data tree to determine and use the respective audio, metadata and video.

3 Entitlement Management Messages

EMMs are delivered through two independent mechanisms. The first delivery method, targeted at two-way devices, shall be through the use of the DOCSIS compliant modem integral to two-way devices. The second delivery method, provided for one-way devices and as an alternate method for two-way devices, is in-band delivery, duplicated on each transport stream. The in-band delivery closely follows the MPEG-2 prescribed method. Details regarding the specifics of each delivery method can be found in the Navigation Data Specification (**Part 2 of this specification**).

A conditional access table (CAT), as defined in ISO/IEC 13818-1 **[1]**, shall be carried in each transport stream and transmitted with the system SI data via DOCSIS delivery using UDP multicast. There are subtle differences between the standard and the proposed system regarding location of the table in the transport stream and these differences are described in detail in Navigation Data Specification (**Part 2 of this specification**). The CAT shall contain one or more CA descriptors, as required by the standard. One of the CA descriptors shall have the CASID, which is defined as 0x0001 for the basic access control system and the table shall provide the EMM PID assignment within a particular transport stream. The EUE shall configure its transport processing elements to receive EMMs carried on this PID. For devices that receive SI through OOB methods such as DOCSIS, the EMM data shall be carried on a multicast IP address and port specified in the CAT message per (**Part 2 of this specification**), but the rest of the process and syntax remains unchanged from the in-band process.

Entitlement Management Messages (EMMs) shall be used to signal code download, trigger emergency alerts, deliver keys and other access control related information. The format, shown in Table 1, conforms to standard MPEG-2 table sections. These different forms of data shall be differentiated and carried within the EMM through the use of unique descriptors, constructed specifically for carrying data contextually related to a particular type of information.

Table 1 – EMM Format

Syntax	No. of Bits	Mnemonic
entitlement_management_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
address_hash	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
address	48	uimsbf
for (i=1; i<N;i++) {		
descriptor()		
}		
CRC_32	32	Rpchof
}		

table_id: An 8-bit field that shall be set to 0x82 to identify this private table uniquely as a unit addressed EMM or to 0x84 to identify this private table uniquely as a group addressed EMM.

section_syntax_indicator: A 1-bit field that shall be set to 0 to identify this table as a private section. The following bit set to zero signifies the data payload as private data.

address_hash: A 16-bit field carrying the hashed value of either the unit or group address, depending upon the ***table_id*** value. The entire 48-bit address field shall be condensed into a single 16-bit value using the 32-bit FNV-1 hash with xor-folding **[**Error! Reference source not found.**]**. This parameter is provided as a method to allow hardware filtering of EMMs and reduce EUE processor workload.

version: A 5 bit field that is monotonically increased for each unique EMM transmitted to a particular addressee or group, as applicable.

address: A 48-bit field carrying either the unit or group address, depending upon the ***table_id*** value. This is the same value used to create the ***address_hash***.

The EMM message carries one or more MPEG-2 format descriptors, which contain the actual entitlement payload. The different possible EMM descriptors are:

Descriptor	Descriptor Tag Value	Implementation
Software Download Command	0xB0	
Signature Data	0xB3	
EAS Message	0xB1	
Fingerprint Message	0xB4	
Program Authorization	0xB5	
Group Assignment	0xB6	
FIPS Assignment	0xB7	
OSD Descriptor	0xB8	
Key Data	0xB2	
Key Index List	0xB9	
Operation Mask	0xBA	
Service Mask	0xBB	
Bouquet	0xBC	
Version	0xBD	
	0xBE	Used for provisioning message IP descriptor
	0xBF	Used for provisioning message DDC descriptor

Every EMM shall contain at least a signature data descriptor, which shall always be the last descriptor in an EMM. The inclusion of any descriptors in addition to the signature data descriptor is optional and there shall be no limitations regarding the number or order of the optional descriptors.

The EMM section is marked as an MPEG-2 private section containing private data. Each EMM section shall not exceed 4096 bytes in length. If the total message payload exceeds 4096 bytes, then multiple sections shall be transmitted and the payload shall span sections, with the ***section_number*** and ***last_section_number*** indicating where a particular section fits in the sequence. In a multi-section EMM, each section shall not be terminated with a 4 byte CRC for that particular section.

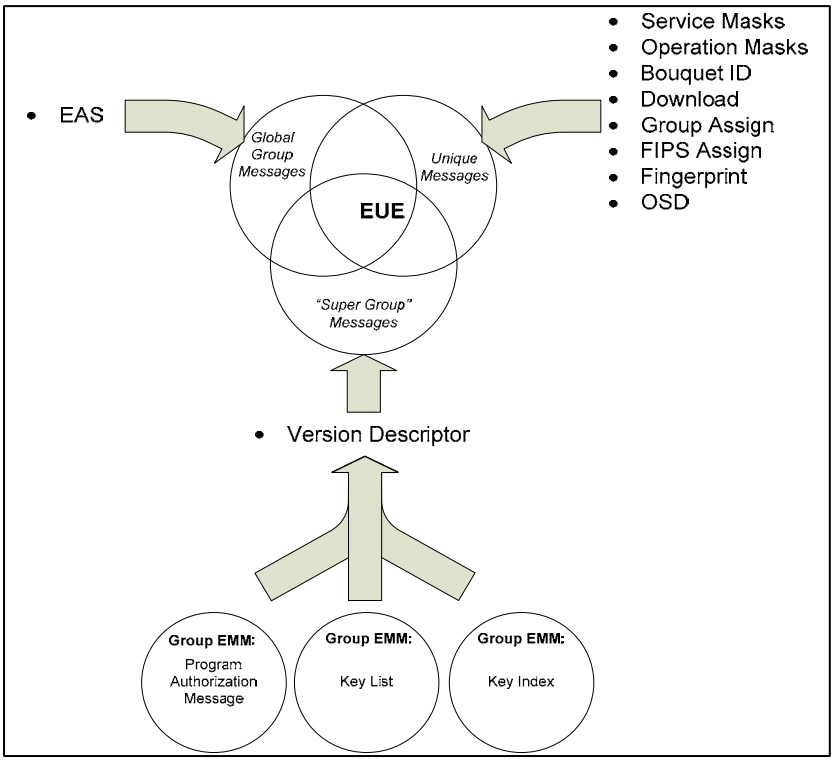


Figure 6 – Group and Message Relationship

The last section in the message shall contain a CRC covering the composite payload, per the EMM section format provided in Table 1. It is the responsibility of the receiving device to buffer a multi-section EMM in the correct sequential order and to remove the intervening section headers before EMM payload concatenation and composite CRC validation.

3.1 EMM Carousels, Carousel Nesting and Group Usage

EMMs are recirculated in virtual carousels based upon address, including group addresses. The period a particular EMM remains in circulation is determined by its importance, relevance, age, available BW, etc. All EMMs in circulation have a unique address and any change in content for an EMM having a particular address is signified by a change version field value associated with each EMM. There shall be only one version of an EMM in circulation on a particular address at any point in time.

3.1.1 Address Priority

Each EUE shall, at a minimum, monitor three different EMM addresses:

- The global EMM address 0x10000FFFFFF.
- An assigned “super group” address, which is assigned by the BAC system during device provisioning.
- Its unique EMM address (EUI-48 MAC address).

While monitoring the three addresses, priority shall be given first to any valid messages carrying the global group EMM address. If there are no valid global EMMs, then secondary priority shall be given to valid message traffic carrying the assigned super group EMM address, if assigned to the EUE. If there is no valid traffic from the global or super groups, then tertiary priority shall be given to any valid message traffic containing the unique EMM address specific to that particular EUE.

3.1.2 EMM Descriptor Hierarchy

EMM descriptors are classified into two categories:

Category I – EMM descriptors having a scope covering multiple endusers

- EAS
- Key List
- Program Authorization Table
- Key Index

Category II – EMM descriptors having a scope covering a single user or EUE

- Operation Mask
- Service Mask
- Bouquet Descriptor
- Software Download Command
- FIPS Assignment
- Group Assignment
- Fingerprint Message
- OSD Descriptor

Category-I descriptors are all large data structures that change relatively infrequently and affect entire nodes, localities or regions. Category-II descriptors can all be combined to fit within an EMM having a single sequence of sections (256 * 4096 byte sections = 1 MByte).

Each EMM containing a single, different category-I descriptor will be carried on continuously on different EMM group addresses, analogous to code download carousels.

It is incumbent upon the EUE to maintain a record in nonvolatile memory of the version number of the last EMM received. This record must be maintained for each group that the has been assigned membership in and is in addition to the records maintained by default for the version number of the last EMM addressed to the EUE unique address and the global group (0x100000FFFFFF). All EMMs received addressed to the EUE or a group in which it has been assign membership in shall have the version number evaluated before further processing. If an incoming EMM has the same version as the last applicable EMM received, the message shall be discarded without further evaluation.

3.1.13 Provisioning Process

Each EUE will initially be provisioned for services through reception of an EMM, containing category-II descriptors, bearing the unique address of the device. This EMM will carry at a minimum a group address assignment and FIPS assignment descriptor. The STB will receive an assignment to a group address carrying the “super group” address applicable to that EUE. The EMM carried in the “super group” typically carries only version and signature descriptors. The version descriptor includes other group addresses pointing to the category-I EMM carousels for key list and program authorization tables appropriate to that EUE based upon service tier, type, region and/or locality.

Refer to Figure 7 for following the process flow. When a EUE receives an EMM, the EUE shall validate the EMM, parse the entire EMM payload and determine whether a group assignment descriptor is part of the payload. The group assignment descriptor is usually carried in a uniquely addressed EMM and is typically intended for assignment of a “super group” to the EUE. Upon determining that a group assignment descriptor

is contained in the EMM payload, the EUE shall defer further processing of the current EMM and will immediately begin monitoring the assigned group address.

If there is an unreceived EMM present on the group address, it must be acted upon first, before processing any remaining descriptors in the previous EMM. Typically, the EMM will be a super group EMM containing a version descriptor. If a version descriptor is present in the validated EMM, it must be processed first before any remaining descriptors are processed. This nesting of EMMs based upon priority forms a recursive process that terminates when all higher priority descriptors/EMMs are exhausted.

Processing of the version descriptor entails checking the version of each category-I descriptor listed in the version table and comparing it to the value stored in EUE memory for the last received and processed version of the EMM carrying that descriptor.

If the version in the EUE is different from the one given in the version descriptor, the EUE shall immediately configure its transport filters to receive the group EMM address for the EMM carrying the descriptor, also indicated in the version descriptor table. No other entries of the version descriptor table shall be processed prior to receiving and processing an EMM occurring on the group address indicated in the current entry of the version descriptor table. If the version cited in the version descriptor is equal to the corresponding entry in the EUE memory, then the next entry in the version descriptor is evaluated. It is important to note that the EUE must process the version descriptor table following the sequence given in the version descriptor, otherwise information may be unintentionally overwritten.

EMMs each containing a single, different category-I descriptor shall be carried continuously on different EMM group addresses, analogous to code download carousels. The EUE will process the EMMs containing these category-I descriptors first, holding any other received category II EMM descriptors for processing until after any identified category-I EMMs have been successfully processed. In this manner, the large descriptor structures may be downloaded by inference through group membership and structure modifiers, like service or operation mask descriptors, can be subsequently applied to the baseline tables received through a group membership.

The resulting EMM architecture is a hierarchy supporting the transmission of only one EMM version on each unique and group address at a time. The MPEG-2 section header version counter is used as intended for a payload change detector and version count overflow merely signifies a version change. It also simplifies system configuration management because large tables like channel mask, key, etc. are only changed in one central place, with the EUE's detecting changes through version incrementing. This schema is very similar to the paradigm used for SI configuration management.

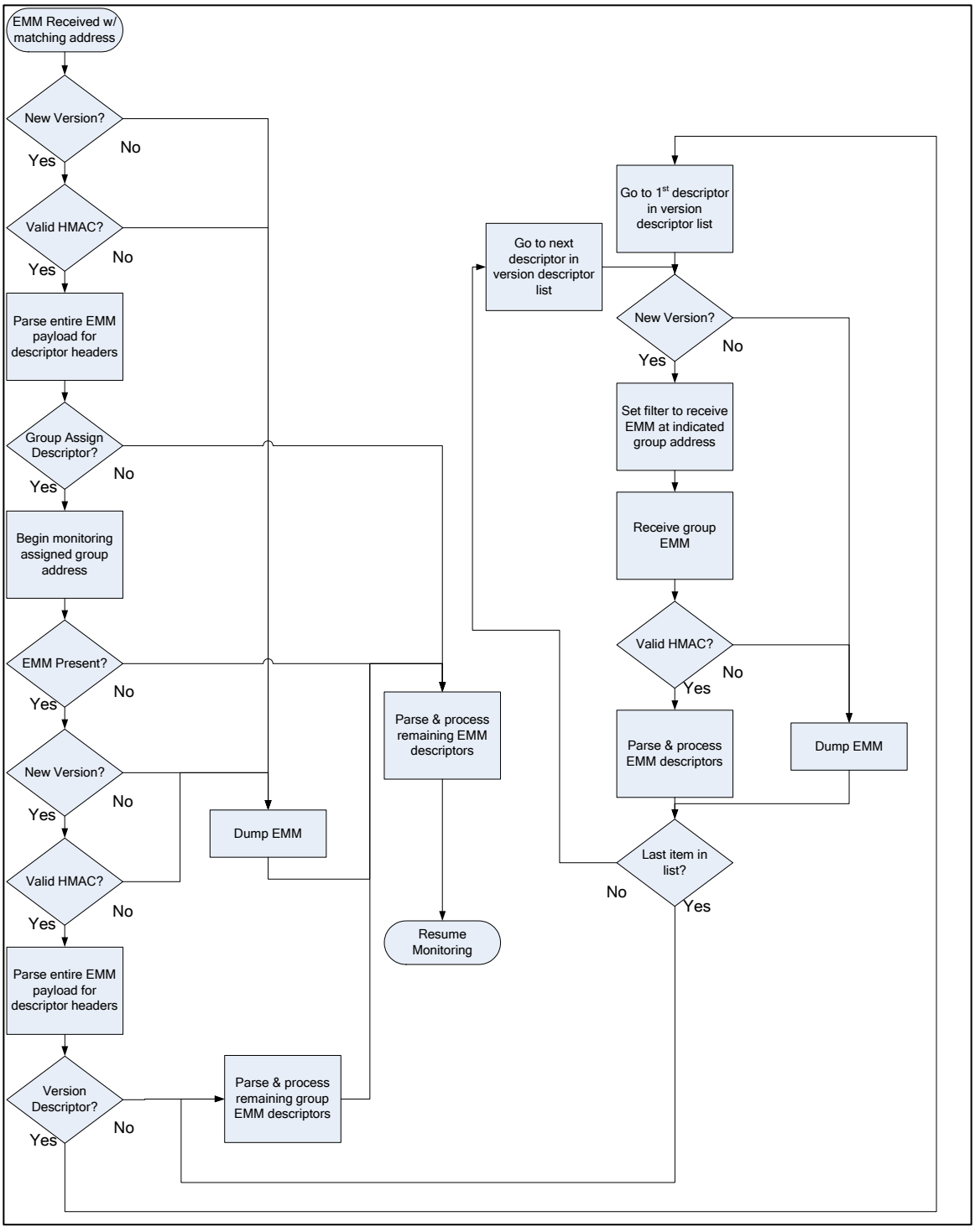


Figure 7 – EMM Processing Flow

3.1.4 Global Group Address Usage

The global group address shall be used to carry EMMs of universal interest to all EUE. EAS messages shall be also transmitted using EMMs carrying the global group address.

It is still possible to send an EMM containing descriptors other than EAS or software download addressed to the global group and the EUE is expected to act upon all valid messages received bearing this address.

3.1.5 Super Group EMM Address Usage

EMMs sent to the super group address to which a particular EUE is assigned are intended to notify the EUE of additional applicable provisioning resources, their current MPEG-2 version and the group EMM address assigned to each of those resources. The notification of the available resources will be accomplished through the use of the version descriptor. This process is how the EUE shall be directed to receive multiple different EMMs, each containing category-I EMM descriptors.

It is still possible to send an EMM containing descriptors other than the version descriptor addressed to the super group assigned to the EUE and the EUE shall act upon all valid messages received bearing this address.

3.1.6 Unique EMM Address Usage

The unique device EMM address is intended for transmission of category-II EMM descriptors to a particular EUE because such descriptors have limited scope, typically affecting a single device or subscriber account. Examples of actions resulting in a uniquely addressed EMM would include resetting, disabling or “brick moding” a EUE, or targeting a software download to a specific device for testing or problem resolution purposes.

It is still possible to send an EMM containing descriptors other than category-II carrying the unique device address and the EUE shall authenticate and act upon all valid messages received.

3.2 EMM Descriptors

The presence of descriptors in a valid EMM not defined in this specification shall not cause any aberrant operation or malfunction of a EUE device. Undefined descriptors contained in an EMM shall be ignored in their entirety and the next descriptor, if present, shall be parsed, if defined and valid. This requirement allows for the addition of new descriptors and the extension of EMM capabilities in support of devices with capabilities beyond the current scope without forcing software changes to all deployed EUE’s. No action shall be taken by the EUE regarding an EMM message until the signature of the entire message has been validated.

3.2.1 Software Download Command Descriptor

The descriptor is a private descriptor providing initiation of a software download to a device. Its format is defined in Table 2.

Table 2 – Software Download Command Descriptor

Syntax	No. of Bits	Identifier
software_download_command_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
download_mode	2	uimsbf
reserved	4	bslbf
download_schedule	2	uimsbf
target_version (Major(16).Mid(8).Minor(8))	32	uimsbf
time_of_change	40	bslbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xB0 to identify this descriptor uniquely as a software download descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

download_mode: A 2-bit field indicating the intended download type to be communicated to the device.

Table 3 – Upgrade Modes for Download Descriptors

Value	*Mode*
0x0	Reserved
0x1	Normal
0x2	Beta upgrade
0x3	Immediate upgrade

download_schedule: A 2-bit field indicating the intended download schedule to be followed by the device.

Table 4 – Upgrade Schedule for Download Descriptors

Value	*Schedule*
0x0	Reboot
0x1	Standby
0x2	Time scheduled upgrade
0x3	Reserved

target_version: This 32-bit field identifies the version number of the upgrade image to be loaded. For **NORMAL** and **BETA** types of upgrades, a value of zero for this field indicates that the EUE should take the highest version available. If the value is non-zero, the EUE should find the specified version and load it. The 32-bit value is sub-defined into Major, Mid and Minor version fields in the following manner:

```
UINT16 majorVersion;
UINT8  midVersion;
UINT8  minorVersion;
```

The version number can be visualized in dot notation as Major.Mid.Minor. This allows version definitions from 0.0.1 to 65535.255.255.

time_of_change: A 40-bit field specifying the date and time of a scheduled download formatted according to reference **[2]**, section 6.2.17. For an **IMMEDIATE** type of upgrade, this field shall be ignored by the device.

3.2.2 EAS Message Descriptor

The EAS message descriptor is a private descriptor providing notification of an impending emergency alert broadcast. Its format is defined in Table 5. The complete EAS message section is encapsulated and carried within the EAS message descriptor.

Table 5 – EAS Message Descriptor

Syntax	No. of Bits	Identifier
EAS_message_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
cable_emergency_alert_section() {		
table_ID	8	uimsbf value 0xD8
section_syntax_indicator	1	'1'
zero	1	'0'
reserved	2	'11'
section_length	12	uimsbf
table_id_extension	16	uimsbf '0x0000'
reserved	2	'11'
sequence_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
protocol_version	8	uimsbf
EAS_Event_ID	16	uimsbf
EAS_originator_code	24	uimsbf three ASCII characters
EAS_event_code_length	8	uimsbf (N)
EAS_event_code	8*N	uimsbf N ASCII characters
nature_of_activation_text_length	8	uimsbf
nature_of_activation_text()	text	uimsbf
alert_message_time_remaining	8	uimsbf seconds range 0..120
event_start_time	32	uimsbf
event_duration	16	uimsbf minutes range 15..6000
reserved	12	bslbf
alert_priority	4	uimsbf
details_OOB_source_ID	16	uimsbf
reserved	6	'111111'
details_major_channel_number	10	uimsbf
reserved	6	'111111'
details_minor_channel_number	10	uimsbf
audio_OOB_source_ID	16	uimsbf
alert_text_length	16	uimsbf
alert_text()	text	
location_code_count		uimsbf range 1..31
for (i=0; i<location_code_count; i++) {		
state_code	8	uimsbf range 0..99
county_subdivision	4	uimsbf range 0..9
reserved	2	'11'
county_code	10	uimsbf range 0..999
}		
exception_count	8	uimsbf
for (i=0; i<exception_count; i++) {		
in_band_reference	1	bslbf
reserved	7	'1111111'
if (in_band_reference) {		
Reserved	6	'111111'
exception_major_channel_number	10	uimsbf
Reserved	6	'111111'
exception_minor_channel_number	10	uimsbf
}		
else {		
reserved	16	bslsbf
exception_OOB_source_ID	16	uimsbf
}		
}		
reserved	6	'111111'
descriptors_length	10	uimsbf
for (i=0; I<N; i++) {		
descriptor()		
}		
CRC_32	32	rpchof
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB1 to identify this descriptor uniquely as an EAS message descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

3.2.3 Key Data Descriptor

The key data descriptor is a private descriptor providing the array of working keys necessary for decrypting content. Its format is defined in Table 6.

Table 6 – Key Data Descriptor

Syntax	No. of Bits	Identifier
key_data_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
key_phase	1	bslbf
reserved	7	bslbf
key_count	16	uimsbf
for (i=0; i<***key_count***;i++) {		
content_key	64	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB2 to identify this descriptor uniquely as a key data descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

key_phase: A single bit field that when set to a '1', indicates that the following array data shall be applied to packets carrying a ***transport_scrambling_control*** value of '11'. If the bit is cleared to a '0', it indicates that the following array data shall be applied to all packets carrying a ***transport_scrambling_control*** value of '10'.

key_count: 16-bit field containing the number of entries in the content key array

content_key: A 64-bit value containing one ETSI-DVB-CSA content key

3.2.4 Signature Data Descriptor

The signature data descriptor is a private descriptor providing the signature payload necessary for authentication of an EMM message. Its format is defined in Table 7.

Table 7 – Signature Data Descriptor

Syntax	No. of Bits	Identifier
signature_data_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
EMM_signature	160	uimsbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xB3 to identify this descriptor uniquely as a signature data descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor. For this particular descriptor, the value is defined to be 0x14.

EMM_signature A 160-bit field carrying the EMM message authentication code using the algorithm defined in section 0.

3.2.5 Fingerprint Message Descriptor

The fingerprint message descriptor is a private descriptor carrying the information necessary to provision and manage the RF fingerprint security function. Its format is defined in Table 8.

Table 8 – Fingerprint Message Descriptor

Syntax	No. of Bits	Identifier
fingerprint_message_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
acclimate_command	1	uimsbf
audit_command	1	uimsbf
reserved	6	uimsbf
challenge_string	16	uimsbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xB4 to identify this descriptor uniquely as a fingerprint message descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

acclimate_command: A 1-bit flag indicating that the EUE should calculate the current fingerprint value and store it in nonvolatile memory.

audit_command: *A* 1-bit flag indicating that an immediate re-calculation of the fingerprint value should be performed and compared with the value stored in the EUE, with the actions specified in section 5.1 taken based upon the outcome of the comparison.

challenge_string: A 16-bit string carrying four 4-bit binary coded decimal (BCD) digits to be rendered using OSD on the EUE.

3.2.6 Program Authorization Descriptor

The program authorization descriptor is a private descriptor carrying the information necessary to provision the EUE to receive program content tiers. Its format is defined in Table 9.

Table 9 – Program Authorization Descriptor

Syntax	No. of Bits	Identifier
program_authorization_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
bouquet_id	16	uimsbf
reserved	16	uimsbf
channel_count	16	uimsbf
for (i=0; i<***channel_count***;i++) {		
logical_channel	16	uimsbf
reserved	16	uimsbf
key_index	16	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB5 to identify this descriptor uniquely as a program authorization descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

bouquet_id: *A* 16-bit field carrying the index to the assigned channel bouquet as defined in the BAT table.

channel_count: A 16-bit field indicating the size of the channel array following this variable.

logical_channel: A 16-bit field specifying a logical channel number. The logical channel identified in this field will be paired to the ***service_mask***, carried in the service_mask descriptor, field as a record pair

The data contained within array shall be stored in EUE nonvolatile memory. In the case where the array has not been received or is an entry for a particular channel otherwise missing, the default value shall be 0x00.

key_index: A 16-bit field specifying a pointer to an element in the data structure, stored in EUE nonvolatile memory containing the content keys for the service. Both the audio and video portions of a given service shall use the same content encryption key.

3.2.7 Group Assignment Descriptor

The group assignment descriptor is a private descriptor carrying the information necessary to associate the EUE with one or more identification groups, enabling the device to respond to group EMMs sent to a group in which the EUE has been assigned membership. Its format is defined in Table 10.

Table 10 – Group Assignment Descriptor

Syntax	No. of Bits	Identifier
Group_assignment_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
group_count	8	uimsbf
for (i=0; i<***group_count***;i++) {		
group_address	48	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB6 to identify this descriptor uniquely as a group assignment descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

group_count: An 8-bit field containing the number of group addresses assigned to the recipient of this message.

group_address A 48-bit field containing a single group address.

3.2.8 FIPS Assignment Descriptor

The FIPS assignment descriptor is a private descriptor carrying the information necessary to associate the EUE with one or more location codes, enabling the device to respond to EAS messages sent concerning a geographic area in which the EUE has been assigned membership. Its format is defined in Table 11.

Table 11 – FIPS Assignment Descriptor

Syntax	No. of Bits	Identifier
FIPS_assignment_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
location_count	8	uimsbf
for (i=0;		
i<***location_count***;i++) {	8	uimsbf
state_code	4	uimsbf
county_subdivision	2	bslbf
reserved	10	uimsbf
county_code		
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB7 to identify this descriptor uniquely as a FIPS assignment descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

location_count: An 8-bit field in the range of 0 to 31 containing the number of geographic location records assigned to the recipient of this message.

state_code: An 8-bit field in the range of 0 to 99 containing the state code assignment for a particular location record.

county_subdivision: A 4-bit field in the range of 0 to 9 containing the county subdivision code assignment for a particular location record.

reserved: A two-bit value carrying the constant '11'.

county_code: A 10-bit field in the range of 0 to 999 containing the county code assignment for a particular location record.

3.2.9 OSD Descriptor

The OSD assignment descriptor is a private descriptor carrying text information that the operator wishes to have displayed on the television screen. Its format is defined in Table 12.

Table 12 – OSD Descriptor

Syntax	No. of Bits	Identifier
OSD_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
text_length	16	uimsbf
for (i=0; i<***text_length***;i++)		
text	text	char
display_mask	16	uimsbf
OSD_start_schedule	40	bslbf
OSD_expiration_schedule	40	bslbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xB8 to identify this descriptor uniquely as an OSD descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

text_length: A 16-bit field indicating the number of characters in the string to be displayed.

display_mask: A 16-bit field indicating the characteristics of the OSD, as defined in Table 13. Values may be logically 'or'ed to create composite cases.

Table 13 – OSD Display Mask Values

Value	*Characteristic*
0x0000	Display immediately/Persistent
0x0001	Extinguish on any user action
0x0002	Extinguish on standby/reboot
0x0004	Extinguish after expiration time
0x0008-00F0	Reserved
0x0100	Display on any user action
0x0200	Display after standby/reboot
0x0400	Display on scheduled start time
0x0800-F000	Reserved

OSD_start_schedule: A 40-bit field specifying the start date and time of a scheduled OSD formatted according to reference **[3]**, section 6.2.17.

OSD_expiration_schedule: A 40-bit field specifying the expiration date and time of a scheduled OSD formatted according to reference **[3]**, section 6.2.17.

3.2.10 Key Index List Descriptor

The key index descriptor is a private descriptor carrying the information necessary to provision the EUE with the relationship between logical channels and content keys and is stored in nonvolatile memory as part of the program authorization table structure. It is a subset of the program authorization descriptor. Its format is defined in Table 14.

Table 14 – Key Index Descriptor

Syntax	No. of Bits	Identifier
key_index_list_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
channel_count	16	uimsbf
for (i=0;		
i<***channel_count***;i++) {	16	uimsbf
key_index		
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xB9 to identify this descriptor uniquely as a key index list descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

channel_count: A 16-bit field indicating the size of the channel array following this variable.

key_index: A 16-bit field specifying a pointer to an element in the data structure, stored in EUE nonvolatile memory containing the content keys for the service. Both the audio and video portions of a given service shall use the same content encryption key.

3.2.11 Operation Mask Descriptor

The operation mask descriptor is a private descriptor carrying the information necessary to update the value of the operation mask, stored in the EUE nonvolatile memory as part of the program authorization table structure. The operation mask alters characteristics of the EUE functionality. It is a subset of the program authorization descriptor. Its format is defined in Table 15.

Table 15 – Operation Mask Descriptor

Syntax	No. of Bits	Identifier
operation_mask_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
operation_mask	16	uimsbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xBA to identify this descriptor uniquely as a operation mask descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

operation_mask: A 16-bit field carrying a binary mask indicating attributes to be applied to the EUE.

Table 16 – Operation Mask Bit Definition

Position	*Mask Bit Definition (Set)*
0x0000	Clears all persistent mask bits
0x0001	Reboot (Non-persistent)
0x0002	Standby (Non-persistent)
0x0004	EUE Deauthorized (Persistent)
0x0008	“Brick Mode” (Persistent)
0xFFF0 to 0x0010	Reserved

Multiple binary mask bits may be combined e.g. 0x0009 would result in a EUE rebooting and remaining in a disabled (brick mode) state after reboot. The data contained within mask shall be stored in EUE nonvolatile memory, except that the two lowest significance bits shall not be evaluated (non-persistent) and their operation shall be a one-time only (immediate) function upon EMM receipt. In the case where the mask has never been received or is otherwise missing, the default value shall be 0x0000.

3.2.12 Service Mask Descriptor

The service mask descriptor is a private descriptor carrying the information necessary to change the attributes associated with the services provisioned in a EUE and stored in nonvolatile memory as part of the program authorization table structure. It is a subset of the program authorization descriptor. Its format is defined in Table 17.

Table 17 – Service Mask Descriptor

Syntax	No. of Bits	Identifier
service_mask_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
item_count	16	uimsbf
for (i=0; i<**item_count**;i++) {		
logical_channel	16	uimsbf
service_mask	16	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xBB to identify this descriptor uniquely as a service mask descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

item_count: A 16-bit field indicating the number of entries in the list following this variable.

logical_channel: A 16-bit field specifying a logical channel number. The logical channel identified in this field will be paired to the subsequent ***service_mask*** field as a record pair

service_mask: A 16-bit field containing a binary mask indicating attributes associated with the corresponding logical channel.

Table 18 – Service Mask Bit Definition

Position	*Mask Bit Definition (Set)*
0x0000	Clears all previous mask bits
0x0001	Channel Unauthorized
0x0002	Macrovision Activated
0xFFFC to 0x0004	Reserved

The data contained within array shall be stored in EUE nonvolatile memory. In the case where the array has not been received or is an entry for a particular channel otherwise missing, the default value shall be 0x00.

3.2.13 Bouquet Descriptor

The BAT descriptor is a private descriptor carrying the information necessary to provision the EUE to receive program content tiers by changing the value of the ***bouquet_id*** variable stored within the EUE nonvolatile memory as part of the program authorization table structure. It is a subset of the program authorization descriptor. Its format is defined in Table 19.

Table 19 – BAT Descriptor

Syntax	No. of Bits	Identifier
bouquet_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
bouquet_id	16	uimsbf
}		

descriptor_tag: An 8-bit field that shall be set to 0xBC to identify this descriptor uniquely as a bouquet descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

bouquet_id: *A* 16-bit field carrying the index to the assigned channel bouquet as defined in the BAT table.

3.2.14 Version Descriptor

The version descriptor is a private descriptor carrying a list of all category-I EMM descriptors and the corresponding version being currently broadcast on the system. Its format is defined in Table 20.

Table 20 – Version Descriptor

Syntax	No. of Bits	Identifier
version_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	32	uimsbf
entry_count	8	uimsbf
for (i=0; i<**entry_count**;i++) {		
emm_descriptor	8	uimsbf
reserved	3	bslbf
current_version	5	uimsbf
group_address	48	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xBD to identify this descriptor uniquely as a version descriptor.

descriptor_length: A 32-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

entry_count: An 8-bit field indicating the number of emm descriptors carried in the table structure following the ***entry_count*** parameter.

emm_descriptor: An 8-bit field specifying a particular category-I EMM descriptor.

current_version: A 5-bit value indicating the currently transmitted MPEG version number of the EMM containing the

group_address: A 48-bit value specifying the group address associated with an EMM carrying the corresponding category-I descriptor.

3.2.15 EMM Signatures

Each EMM shall be validated after decryption and prior to implementation by the EUE. The process of validation employs use of the HMAC algorithm upon the SHA-1 hash of the entire decrypted EMM, including all embedded descriptors. The scope of the signature starts at the most significant bit of the table_id of the EMM header and extends up to, and includes the least significant bit of the last descriptor other than the signature descriptor. The signature descriptor shall be calculated and appended to the proto-EMM and is always the last descriptor in the section. The scope of the EMM section CRC shall include all components of the EMM section, including the signature descriptor. The 160-bit HMAC calculated by the EUE value shall be compared against the value contained within the signature descriptor. If the values exactly match, the entire message shall be considered legitimate. If there is not a match, the entire message shall be disregarded.

The EMM HMAC key value is a 512-bit constant that shall be securely embedded in the EUE software application code image. The EMM HMAC key value (hexadecimal) is defined by the application.

4 Software Download

NOTE – Throughout the chapter on software download, references will be made to the EUE upgrade image containing multiple, discrete files. While this reference is made to accommodate EUE manufacturers that may choose to use a file-based organization of data within FLASH, it should in no way be construed as implying that a file-based system is required or expected. The term “file” may be used interchangeably with the terms: module, structure, element, component, library, etc. in devices using other forms of logical organization and possibly containing multiple data components to comprise a software download image.

It is equally acceptable for a device to have a single monolithic image with no structural modularity whatsoever, the philosophic discussion of the advantages or disadvantages of each being outside the scope of this document.

This section defines the SBS method (Slow/Simple But Sure) for implementing remote, fail-safe software upgrading of EUE devices. The SBS upgrade process will be implemented by SBS-Client software, residing in the EUE. The purpose of the SBS-Client is twofold:

1. Provide a fail-safe upgrade method for the EUE bootrom program.

2. Provide an upgrade method to newer software versions in the EUE.

On the EUE device, the upgrade system consists of two components:

1. An SBS-Client resident in the ROM based bootrom, accessible only upon reboot of the device.

2. An upgrade agent application, resident in the application software space. This application shall be functional during normal operation of the device.

The SBS-Client shall be hardcoded into the inerasable bootrom memory, thereby always providing an upgrade method should there be a failure or corruption of the main FLASH memory. The SBS-Client is the engine that actually performs software download, authentication and upgrade to the EUE device FLASH memory. It operates in two modes:

1. Autonomous upgrade at boot time, determined through evaluation of the data download carousel (DDC) message data, part of the provisioning messages carried in-band on all overlay transports on PID 0x1F00.

2. Directed upgrade, based upon a trigger message received from the headend/source via EMM, which is interpreted by the upgrade agent. The upgrade agent in-turn places a semaphore for the SBS-Client containing specific upgrade instructions based upon the received EMM in nonvolatile memory, then initiating a reboot of the device to activate the SBS-Client in order to perform the actual upgrade action.

The second supporting portion of the upgrade system, known as the upgrade agent, shall reside in the erasable FLASH and this shall be callable by the higher-level application code. The SBS-Client, in conjunction with the upgrade agent module, can be used to provide a more enhanced/managed upgrade scheme. The headend/source is responsible for notifying the upgrade agent regarding the availability of upgrades and the agent determines the upgrade action to follow based on the indicated upgrade type. However, it is imperative that regardless of the higher level upgrade schemes employed that the bootrom resident SBS-Client should always be capable of performing an upgrade based on information contained within the in-band upgrade data stream alone.

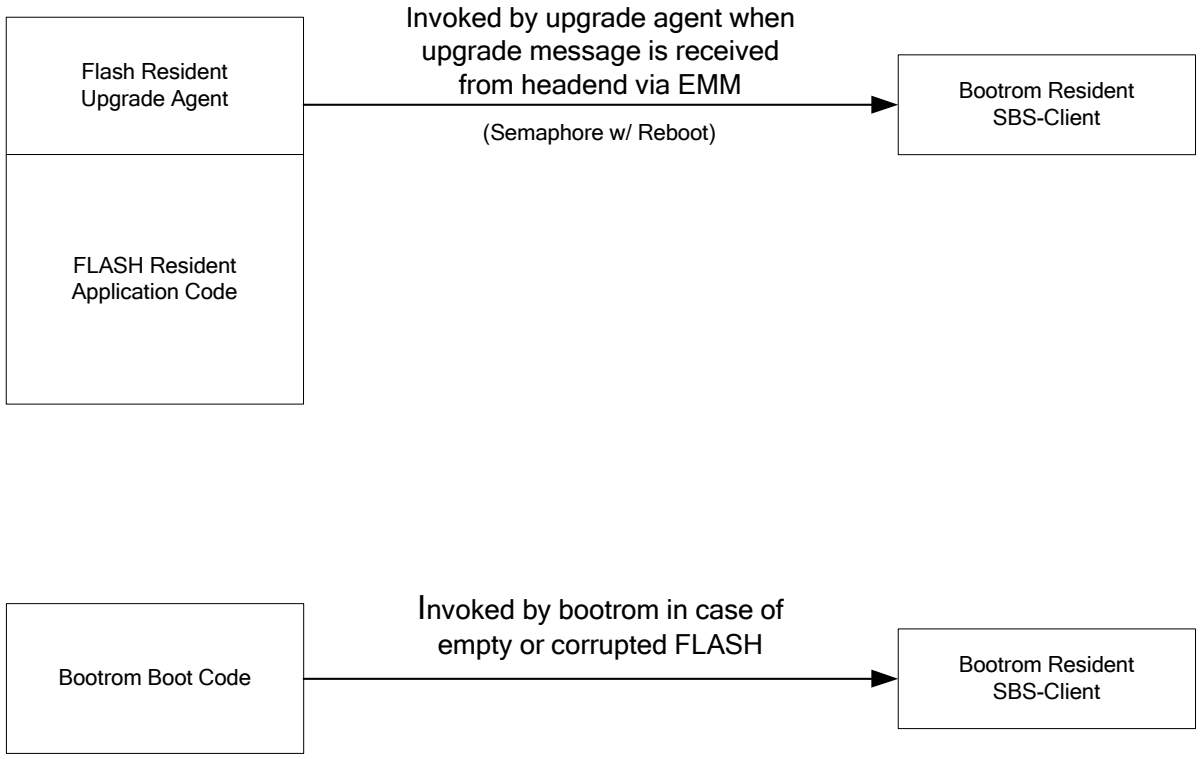


Figure 8 – SBS-Client Invocation

There are three basic scenarios where the SBS-Client may be invoked:

1. An empty or corrupted FLASH is detected by the bootrom image version/corruption checking functions, requiring an immediate upgrade to the latest available code for that particular make/model of EUE.

2. The upgrade agent, responsible for interfacing with the headend/source to receive information/commands regarding required upgrades. It is assumed that the upgrade agent will receive upgrade notification messages from the headend/source and shall then be responsible for deciding how to perform that upgrade. The upgrade mode of that particular version will determine how and when the upgrade is performed. The basic upgrade modes may be:

- **Immediate upgrade** The EUE must unconditionally upgrade its code immediately.
- **Normal upgrade** The EUE shall perform the upgrade based upon the occurrence of one of two events:

  a. The criteria established in the download EMM **download_schedule** variable is met.

  b. The EUE is rebooted through external or other means prior to it meeting the criteria in (a), above.
- **Beta upgrade** The EUE will be pre-selected as part of a beta group to allow upgrading of this code.

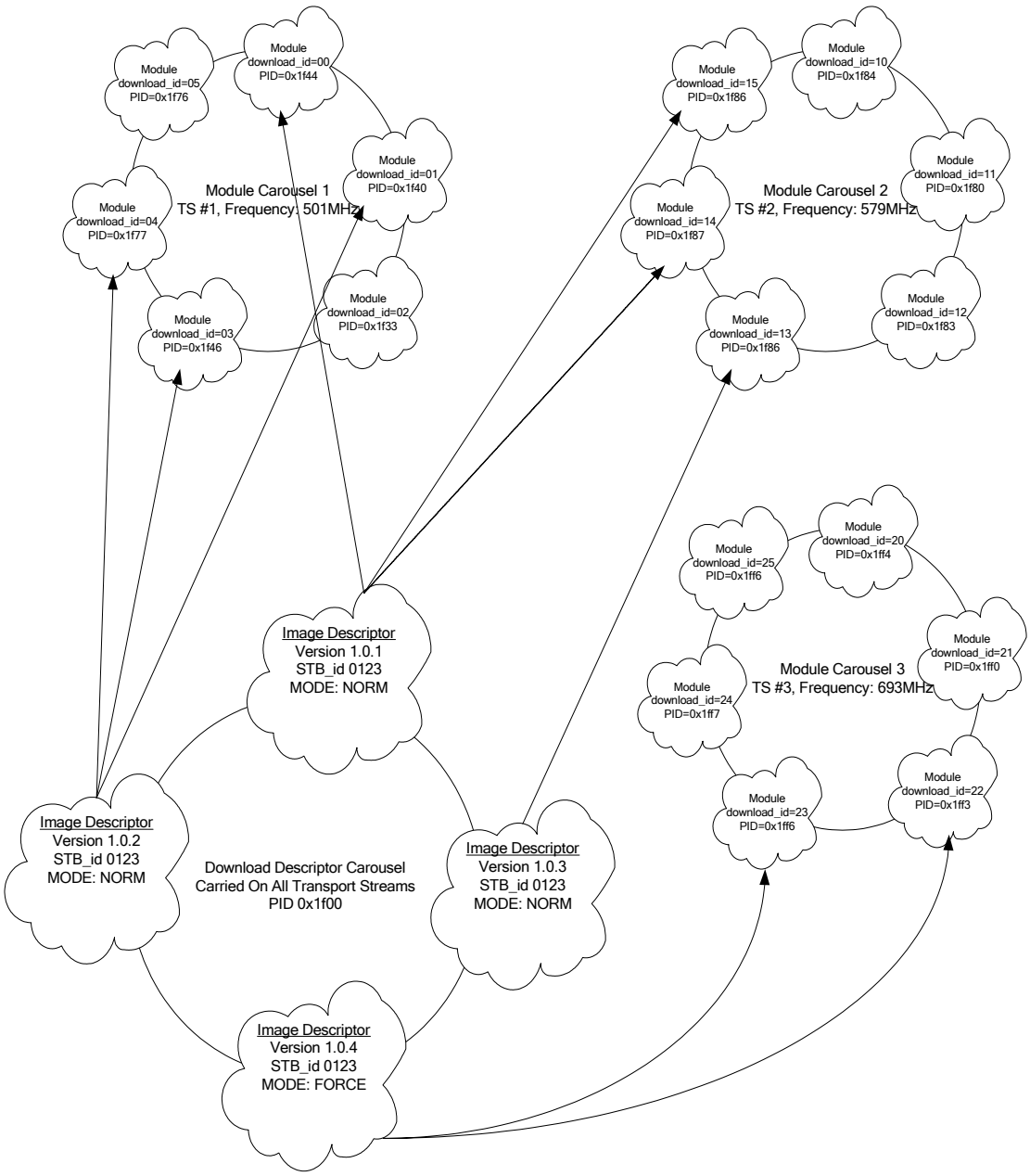
By using various triggers in the upgrade notification message, these modes can be extended to provide much broader functionality.

The in-band upgrade data shall be encapsulated by the module delivery protocol, defined within this document and the protocol-encapsulated data shall be carried in the MPEG-2 Transport as Private Data within Transport Stream Packets, as defined by *ISO/IEC 13818-1, Annex H, H.0, Private Data, Transport Stream packet Table 2-2* **[1]**. This is also compatible with the DVB standard for Data Piping defined in *EN 301 192, ETSI-DVB Specification for Data Broadcasting, Data Piping* **[6]**.

All information describing the images available in download data carousels shall be contained in a download data carousel (DDC) message, which is a descriptor that resides within the provisioning message table, an “MPEG-like” section delivered to convey system specific configuration information to a EUE attempting to boot and access network resources. The EUE will reference the DDC message to determine the available images and the associated resources. The provisioning message and its associated DDC descriptor is a low data content, low data rate message carousel and shall be carried on each transport stream on a single well-known PID. The EUE shall have a method for viewing/entering the provisioning message PID from the remote control/television display, allowing the default PID value set at the factory to be overwritten. Once entered, the value shall be stored in nonvolatile memory and used as the provisioning message PID until superseded by manual entry of another value.

The DDC message contains indices providing access to various upgrade images, which may be distributed arbitrarily across multiple PIDs on multiple transport streams. The module delivery carousels containing components of an upgrade image are located via information contained in the download descriptors contained within the DDC message. Any upgrade image may consist of multiple modules at various locations.

Module Object/Download Descriptor Relationship Example for
Software Upgrade Carousels



Module delivery carousels may contain multiple versions of code destined for different models of EUE. The code images shall be uniquely identifiable and shall be continuously transmitted in a data carousel. It is the responsibility of the EUE to determine and receive the correct version for that particular model through use of the download descriptor messages.

Figure 9 – Module/Descriptor Relationship

In the case of an empty or corrupted EUE FLASH memory, the SBS-Client residing in the bootrom code shall not be allowed to download beta versions of code applicable to that model. It shall be capable of upgrading only to the highest applicable (non-beta) code version available. Beta images may only be accepted by the SBS-Client if so commanded by the upgrade agent triggered by upgrade commands from the headend/source.

An upgrade image consists of one or more module images. The module image data will consist of a set of files or components that are packed into the module image. The module image shall contain enough information to restore files from the packed image correctly. The components, files and scripts constituting an upgrade/module shall be delivered to the SBS-Server, resident at the headend/source.

Each module image shall be assigned to a unique transport frequency/PID combination and shall be a private data stream segmented into multiple “MPEG-like” sections. The maximum size of any module section shall be 4096 bytes. Each section shall be encapsulated within the module delivery protocol. Each section shall be further segmented into MPEG-2 transport packets. The transport packet containing the start of a EUE upgrade protocol section shall have its Payload Unit Start Indicator (PUSI) set to one; all other packets shall have the PUSI set to zero.

All code images shall have a unique download ID, which shall be valid during the time that the code image is available. Two separate versions of code applicable to the same EUE shall have both different download IDs and different version information. It shall be that as soon as the EUE identifies the version and download ID of the applicable code image, reception of module delivery protocol sections can begin in any order. However, the EUE is responsible for ensuring that all sections are correctly received and verified before attempting to apply the upgrade.

The module delivery protocol allows for a protocol identification field so that future implementation of different protocols shall be possible. The EUE shall check the protocol version of each section and shall ignore all sections that are coded with incompatible protocols.

Each module section shall contain a header containing information indicating the protocol version, target model, download ID, module ID, module version information, the total number of sections and the section number assigned to a particular section. The header also contains a CRC that allows confirmation of the validity of a particular header. Sections comprising a module shall be sequentially numbered such that when arranged in order, the section headers removed and concatenated the original packed image shall be reconstituted. In addition to the header and image payload data, each section contains a cryptographic signature (HMAC), calculated over the entire section structure including the header. The signature allows determination by the EUE of both integrity and validity of received module sections in 4096 byte segments prior to storage in FLASH memory.

While the sections of a module shall always be transmitted sequentially in order, the EUE does not need to wait for the first section in a module to start the download process. Because of the segmentation process and the data contained in each section header, it is possible to begin receiving sections out of sequence and re-order them in EUE memory or flash after signature validation. Likewise, if a particular section fails validation, it is not necessary to suspend the download process and wait for the next transmission of a particular section. Because the section size is transmitted in the section header prior to image payload data and the header carries a CRC to validate integrity, it is possible to validate a header and reserve room for the image payload that follows based upon the actual section size indicated in the header. If upon receipt of the data and signature validation it is found that there is a signature mismatch, subsequent sections may still be loaded and the errant section data replaced when the carousel next transmits the data. If the EUE uses a FIFO structure for buffering received sections comprising a module, it is possible for the EUE to re-order sections received without pausing for the start of the carousel when starting the download of a module.

A version/release specific file shall also be generated which shall identify the version information, applicable EUE models and any other information required by the SBS-Server to determine how the image should be packed and encapsulated before insertion in the transport stream.

4.1 EUE Device Upgrade Process

The bootloader upgrade process and policies defined in this section require the following supporting elements:

- Headend/source signature server
- Headend/source in-band DDC message server
- Headend/source in-band download carousel server
- EUE resident signature client
- EUE resident upgrade agent
- EUE resident upgrade process in bootloader

4.1.1 Headend/Source Signature Server

All images presented to the EUE will be signed at the headend/source before being put on the download carousel. The signing process is performed within the key generator/manager server at the headend/source, the input being a specially formatted image file created from a EUE device executable image received from the EUE manufacturer. The output of this process is a signed and specially formatted file, which is converted into carousel compatible binary format. The signature is defined as the 160-byte result from the application of the HMAC algorithm **[9]**, using the 512 bit download HMAC key defined by the application, upon the calculated SHA-1 **[8]** value of the module.

4.1.2 EUE Resident Signature Client

To verify proper signature of any image or descriptor the EUE must use the bootloader or hardware based signature client. This small client is an application of the HMAC standard **[9]**, using a 512-bit constant for the HMAC key value that shall be securely embedded in the EUE bootrom code image. The download HMAC key value (hexadecimal) is defined uniquely for each application.

The signature is defined as the 160-byte result from the application of the HMAC algorithm, using the documented 512 bit download HMAC key, upon the calculated SHA-1 **[8]** value of the module.

4.1.3 EUE Resident Upgrade Agent – *push_command_set* Interface

The upgrade agent is a module that runs in the application space of the EUE. Its job is to process all upgrade messages from the headend/source and coordinate the upgrade process according to criteria within the message. In the event that a validated message is received from the headend/source indicating that an upgrade action is to be taken, a semaphore known as a ***push_command_set*** is placed in nonvolatile memory by the upgrade agent containing the relevant information necessary for the SBS-Client to complete the intended action. When the SBS-Client application gains control of the EUE during the next boot cycle, (which may be as a direct result of the upgrade agent forcing a reboot in response to a headend/source message indicating immediate upgrade), the SBS-Client looks for the presence of a ***push_command_set*** to determine whether the headend/source intended a specific upgrade action be taken.

4.1.4 Headend/source Carousel Support for FORCED, NORMAL and BETA Images

The processes described rely on tags defined in the carousel images. These tags define the type of image that is in the carousel. The tags are inserted before the image is presented to the key generator server for signature.

4.2 Upgrade Image Types and Policies

The EUE shall support receiving the following image types:

1. **FORCED** image
2. **NORMAL** image
3. **BETA** image

Each download image descriptor shall be marked with one of these modes, when it is prepared for placement on the upgrade carousel.

The following policies apply to images in general:

1. **FORCED** images should be used only when immediate upgrade of all EUEs is critically required.
2. **NORMAL** image is the default type for upgrade images.
3. At least one **NORMAL** image should exist on the download carousel for each EUE type.
4. Use of **BETA** images is optional.

4.2.1 FORCED Image Type

This tag defines the image as a **FORCED** upgrade image. If such an image is present on a carousel, the EUE must accept that image above all other images, if and only if, the **FORCED** image version differs from the currently installed image.

The following policies apply to **FORCED** Images:

1. Only one **FORCED** image per EUE type may exist at any one time.
2. The EUE will only accept a **FORCED** image if its version differs from currently installed version.
3. The EUE will always accept **FORCED** image even if it was commanded to accept another **NORMAL** image.

4.2.2 NORMAL Image Type

This tag defines the associated image as a normal upgrade image on the carousel. For a given EUE type, multiple **NORMAL** images may coexist on the carousel and shall be differentiated by the assigned image version number. This should be the normal upgrade method, where the EUE is commanded to accept any version of **NORMAL** image via commands from the headend/source.

The following policies apply to **NORMAL** images:

1. Multiple **NORMAL** images, differentiated by version number may coexist on a download carousel.
2. The EUE can be ordered to upgrade to lower, same or higher **NORMAL** image versions.
3. In the absence of a **FORCED** image, a EUE with corrupted or empty FLASH shall attempt to accept the highest available version of **NORMAL** image.

4.2.3 BETA Image Type

This tag defines the associated image as a special type, which should only be accepted by EUE selected as part of a beta program. The main purpose of defining a **BETA** image type is to prevent the general EUE population from accepting a **BETA** image, if that **BETA** image were the highest version available and the EUE was attempting to accept the highest available image version. In other words, a EUE will never accept a

BETA image as a default image. It must be specifically commanded to accept the **BETA** image via the headend/source.

The following policies apply to **BETA** images:

1. Multiple **BETA** images, differentiated by the image version numbers, may coexist on a carousel.
2. The EUE can be ordered to upgrade to lower, same or higher BETA image versions.
3. In absence of **FORCED** image, a corrupt or empty EUE shall not attempt to accept the highest available version of a **BETA** image.

4.3 Upgrade Process Overview

The EUE shall support both push and pull models for upgrading.

4.3.1 Pull Software Upgrade Process

The EUE will pull an upgrade from the headend/source under the following conditions:

1. No detectable version exists on the EUE.
2. The currently installed version has CRC32 errors in certain critical files.

On boot-up, the SBS-Client in the bootloader application checks the FLASH memory for the presence of a valid working application image. If this image exists, an integrity check of the critical components within the image is performed. If the image is either nonexistent or corrupted, the EUE shall attempt to pull a version from the headend/source.

The pull is initiated by the SBS-Client accessing the DDC message descriptor, contained within the provisioning message, to find a download image applicable to the particular EUE type. From the DDC message, the SBS-Client can determine the location of the module(s) of the selected image. The SBS-Client then tunes and/or configures the transport demux filter to access the indicated frequency/PID for the desired modules comprising the upgrade image and starts the upgrade process.

4.3.2 Push Software Upgrade Process

The push model is used by the headend/source to force a EUE to upgrade to a particular version of software. A push upgrade can be initiated in one of two ways:

1. A command from the headend/source via an EMM to the EUE upgrade agent.
2. Special marking of an image referenced in the DDC message.

Method 1, above, requires that the EUE be powered up and operational to receive the command. The result of a headend/source push command is a ***push_command_set***, which is a semaphore stored by the upgrade agent in nonvolatile memory. It contains a set of commands that are presented to the SBS-Client in the bootloader for action upon the next device boot-up. A reboot of the device may be invoked directly by the upgrade agent depending upon urgency of the upgrade command from the headend/source.

Method 2, above, is a way of forcing all EUEs of a particular type to receive an upgrade upon next boot-up. The EUE can only receive this message upon boot-up therefore boxes that remain continuously powered on will not receive this message. This method should be used with care as it supersedes all other upgrade modes. For example, if a EUE is part of a beta code release group and the **FORCED** tag is set on an image, then that EUE will accept the **FORCED** code image over the **BETA** code image. This mode is intended for the situation where every EUE of a particular type must be **FORCED** to a specific software version.

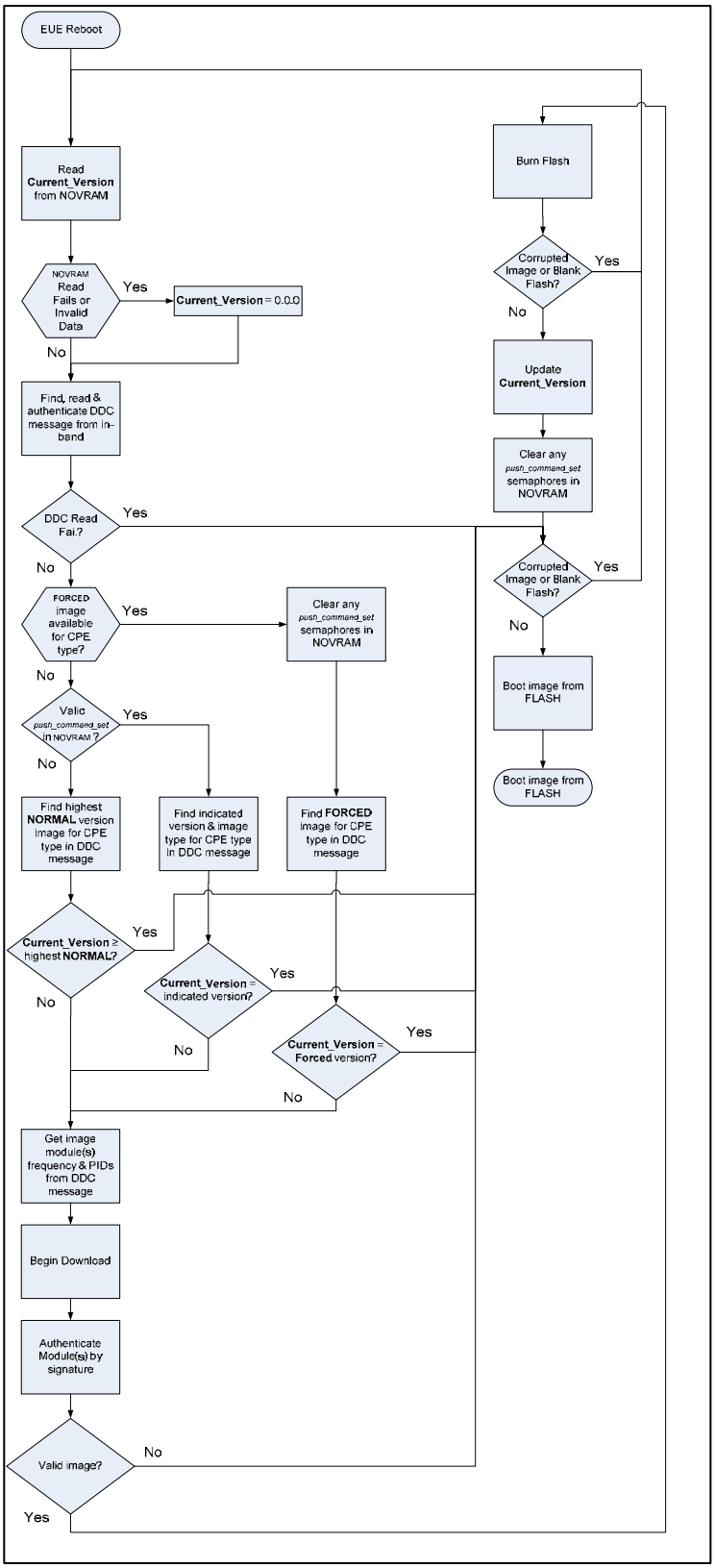


Figure 10 – SBS-Client Upgrade Logic

4.3.3 Bootloader Upgrade Detection Process & Policies

The bootloader upgrade detection process, using the resident SBS-Client, is only executed at boot time. The process, illustrated in Figure 10 , takes the following steps:

1. Check for currently installed version label
2. Check download descriptor carousel (DDC) for FORCED version
3. Check for push_command_set from headend/source via the upgrade agent
4. Check current version for corruption (uses CRC32 check of critical components)
5. Boot current version

Each of the steps above has certain associated policies, which determine the route taken in the event of failure or error conditions described in detail later in this document.

However, this process may be circumvented by certain failures or error conditions that arise during the execution of these steps.

4.3.3.1 Current Version Checking

This step attempts to determine what version of software is currently installed on the EUE. If no version is detected, using a method specific to the EUE determined by the EUE manufacturer, then the internal variable, ***CURRENT_VERSION***, is set to 0.0.0.

The overall policy is to get the EUE operational with whatever applicable code it can find. In the event that the version check fails, the following policy is used in order of priority to determine the version to accept:

1. Accept **FORCED** version listed in the DDC for the specific EUE model.
2. Accept the ***push_command_set*** specified version for the specific EUE model.
3. Accept the highest applicable version for the specific EUE model.

In the event that a **FORCED** version is available according to the DDC message, the EUE will accept that version. A **FORCED** version always supersedes all other versions and in this case, the EUE shall ignore and remove any outstanding ***push_command_set*** requests.

If no **FORCED** version is available, the EUE will check if a ***push_command_set*** is resident in nonvolatile memory. If so, the EUE shall attempt to accept the version defined in the ***push_command_set***. If the ***push_command_set*** is successfully processed, then the request is subsequently removed after execution is completed and confirmed.

If the ***push_command_set*** is not successfully processed, the ***push_command_set*** request is not removed and the flow continues to the next step. The ***push_command_set*** request is not removed in this case because it is a command from the headend/source to push a particular version onto the EUE. Until that command is removed or overridden, the EUE should attempt on each boot-up to accept the specified version.

If the version defined in the ***push_command_set*** is not found or a ***push_command_set*** request is not available, the EUE will attempt to accept the highest **NORMAL** available version.

An important distinction to note is that carousel images may be tagged as in the DDC as **BETA** images. These images are special in that they cannot be accepted by the EUE through any method other than through the ***push_command_set***. In other words, if a **BETA** image is the highest available image, then the EUE shall not accept it as highest and must take a lower versioned **NORMAL** image as the highest available image unless specifically commanded through a ***push_command_set*** message to take a **BETA** image.

If no version is available, the EUE will continually loop in this step until a valid upgrade has completed.

4.3.3.2 FORCED Version Checking

This process determines if there is a **FORCED** version available in the carousel, per the DDC message. A **FORCED** image always supersedes all other image types and a EUE must accept a **FORCED** image if its version is different from the currently installed version. Below are the steps of the **FORCED** image check:

1. Examine the DDC message and detect if a **FORCED** image exists
2. If the **FORCED** version is not equal to the ***CURRENT_VERSION,*** then accept the **FORCED** version
3. Remove any outstanding ***push_command_set*** requests because FORCED is the highest priority upgrade type.
4. If the **FORCED** version is equal to the ***CURRENT_VERSION***, then ignore any outstanding ***push_command_set*** requests and boot with the current version.
5. If no **FORCED** version exists, proceed to the next step.

4.3.3.3 *push_command_set* Checking

This process checks for outstanding ***push_command_set*** requests.

A ***push_command_set*** request is a message created by the upgrade agent application and stored in nonvolatile memory in response to an EMM message addressed to either an individual EUE device or group of devices from the headend/source. The ***push_command_set*** supports a ***version_info*** field to indicate the appropriate version to access on the upgrade carousel.

4.3.3.3.1 Push Command *version_info*

This field defines the version of the requested upgrade and must be defined, compliant with the following policies:

1. A version of 0.0.0 is interpreted as “take the highest applicable **NORMAL** image available”.
2. The version number does not specify whether the image is intended for **BETA** users. It is up to the EMM addressing scheme to determine that EUEs within a beta group get messages in the correct manner. The EUE must check the ***beta_flag*** field, also carried within the ***push_command_set*** message, to determine if a beta version is the intended upgrade image.
3. If the version defined is same as currently installed version, the EUE will re-attempt to get the current image. This facilitates forced re-upgrade of same version.
4. The ***push_command_set*** request is removed after successful completion.
5. In the event of unsuccessful completion, the EUE will boot with the currently installed version, the ***push_command_set*** request is retained and re-processed at next boot time.

4.3.4 Image Corruption Checking

This step checks various portions of the image in an attempt to detect if those portions have been corrupted since last boot. An image can be defined with some, none or all of its components marked for checking. The checking takes place at each boot time, immediately before the image is loaded.

If the image is determined to be corrupted, then the ***CURRENT_VERSION*** global variable is set to 0.0.0 and the upgrade process is invoked. In this state, the EUE is considered to be in a non-usable state and a “get whatever is available” policy is applied, as defined in Section 0.

If the image passes the checks, then the bootloader will load the FLASH image into SDRAM and proceed with execution from the memory space containing the image copy.

4.4 Upgrade Scenarios

The following upgrade scenarios have been defined to illustrate how the bootloader upgrade process shall deal with:

1. Production line, blank EUE FLASH
2. EUE shipped from factory to customer
3. EUE image has been corrupted
4. Immediate case, all EUEs must be upgraded to a particular version ASAP
5. Normal release upgrade
6. Beta users upgrade

Most of these scenarios also require headend/source message delivery to the EUE via EMM, described elsewhere in this document.

4.4.1 EUE Delivered to a Customer/Enduser

When a EUE is delivered to customer/enduser site, the EUE may need to be upgraded to the latest production version, if not done so at the warehouse prior to delivery. To facilitate this, the factory shall create a ***push_command_set*** request with the version set to version 0.0.0 (take highest available) prior to shipping the EUE to the MSO. In such a case, when the EUE is powered up for the first time at customer/enduser site or MSO warehouse, it will perform an upgrade. If the carousel is not operational, then the EUE will continue using whatever factory installed software version exists in system FLASH until the EUE is rebooted and a local version can be successfully accepted.

4.4.2 EUE Image Corruption

If for any reason the bootloader process determines the EUE current image is missing (blank box) or otherwise found to be corrupted, it will attempt to accept whatever non-beta version it can in order to become operational again.

This detection only happens at boot time and if no version is available or if the carousel is not operational, the EUE will remain in a searching state and be unusable to the customer. If the EUE can accept a version, then the EUE will become usable again after downloading the image.

Once the EUE has become operational, the headend/source is responsible for managing the upgrading of the EUE in the normal manner.

4.4.3 Critical/Immediate Upgrade

This scenario can be achieved by a combination of the two push methods. Tagging the image as **FORCED** will cause currently powered off EUEs to automatically download the upgrade upon next boot up. Sending a ***push_command_set*** request with some urgency criteria via EMM will immediately force EUEs that are already powered to attempt the upgrade.

This scenario should only be used when it is imperative that all or as many possible EUEs are upgraded with the greatest urgency. Once there is confidence that sufficient EUEs have been **FORCED** upgraded, the carousel should be reverted to the normal upgrade method.

4.4.4 Normal Release Upgrade

The normal course of MSO operations might involve adding a new **NORMAL** image to the carousel. By using various messaging and timing criteria between headend/source and upgrade agent in the EUE, the upgrade

can proceed in an organized and controlled manner. With this process, the upgrade agent will typically determine the time and version of upgrade based upon the information passed down from the headend/source via EMM. The upgrade agent shall create a set of conditions that, when triggered, initiate an upgrade.

INFORMATIVE NOTE – Typically, this might include scenarios such as:

- Upgrade to version 1.2.3 upon next Power-up.
- Upgrade to version 1.2.3 upon next Stand-by.
- Upgrade to version 1.2.3 after Midnight March 31st 2006.

These types of triggers/conditions are defined between the headend/source and the upgrade agent. It is sufficient to say that as far as the bootloader upgrade process is concerned, any set of conditions or triggers results in a ***push_command_set*** that will be executed on boot up. It is up to the upgrade agent to determine the timing of initiating a reboot and the generation of the ***push_command_set*** request.

4.4.5 Beta Users Upgrade

Beta user upgrades are handled in the same manner as normal upgrades. It is up to the EMM addressing scheme to determine which EUEs belong to any particular beta group. The EUE has no concept of beta groups other than the fact that there may be images in the carousel that are tagged as **BETA** images. The EUE shall only accept **BETA** images when commanded via a ***push_command_set*** request though the ***beta_flag*** variable and shall not accept a **BETA** image when it is in a "find any image to boot" mode, such as encountered in the case of a blank or corrupted FLASH memory.

4.5 Provisioning Messages

Provisioning messages are mapped into MPEG-2 Transport packet payloads. The PUSI bit of the MPEG Packet header being set indicates the start of a provisioning message. Subsequent packets on the same PID contain the remainder of the message. One message must complete before another may start. The provisioning message carries the basic information necessary for a EUE device, either one-way or two-way, to boot.

The provisioning message is carried as a private data stream and segmented into "MPEG-like" sections. Each provisioning message section shall not exceed 4096 bytes in length. If the total message payload exceeds 4096 bytes, then multiple sections shall be transmitted and the payload shall span sections, with the **section_number** and **last_section_number** indicating where a particular section fits in the sequence. In a multi-section provisioning message, each section does not terminate in a 4-byte CRC for each particular section.

Since descriptors defined in the provisioning message format each contain a signature or CRC, each descriptor (and message header) carries its own validation data internally and there is no composite CRC for the provisioning message. It is the responsibility of the receiving device to buffer a multi-section provisioning message in the correct sequential order and to remove any intervening section headers before provisioning message payload concatenation and descriptor CRC or signature validation. Because of the small data set sizes associated with the provisioning message, it is unlikely that a message would span many sections and the additional complexity of intermediate CRCs is therefore unwarranted. If an unexpected section header is encountered before reaching the data set size provided in ***section_length*** field during parsing of a received section in a multi-section message, the errant section should be discarded and re-received. If a descriptor fails to validate through evaluation of the signature or CRC, as appropriate, then the entire message should be re-received and the errant descriptor re-evaluated.

Table 21 – Provisioning Message Format

Syntax	No. of Bits	Mnemonic
provisioning_message_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
protocol_version	8	uimsbf
message_version	8	uimsbf
SI_provisioning_header(){		
reserved	3	bslbf
NIT_PID	13	uimsbf
reserved	3	bslbf
BAT_PID	13	uimsbf
reserved	3	bslbf
SDT_PID	13	uimsbf
reserved	3	bslbf
TOT_PID	13	uimsbf
reserved	3	bslbf
CAT_PID	13	uimsbf
header_version_number	8	uimsbf
}		
reserved	7	bslbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
CRC_32	32	rpchof
for (i=1; i<N;i++) {		
descriptor()		
}		
}		

table_id: An 8-bit field that shall be set to **0x86** to identify this private data table uniquely as a provisioning message.

section_syntax_indicator: A 1-bit field that shall be set to 0 to identify this table as a private section.

private_indicator: A 1-bit field that shall be set to 0 to identify the section payload as private data.

section_length: A 12-bit field that indicates the length of the entire section, with the count starting immediately following the ***section_length*** field, including all descriptors.

protocol_version: An 8-bit field indicating the version of the provision message section protocol.

message_version: An 8-bit field indicating the version of the provision message section. Each successively issued version of the section shall be unique and monotonically increasing.

header_version_number: An 8-bit field that shall that contains the version number of the ***SI_provisioning_header*** portion of the section. It shall be incremented by one whenever a change in the information carried within the header portion occurs.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in **[1]**, annex B after processing the entire span between the start of the table and the ***last_section_number***, inclusive.

The provisioning message will always carry at least one and possibly additional descriptors, which contain the actual provisioning payload. The different possible provisioning descriptors are:

Descriptor	Descriptor Tag Value
C Message (Always present)	0xBF
Provisioning	0xBE

4.5.1 Two-Way IP Provisioning Descriptor

The two-way IP provisioning descriptor is a private descriptor providing a two-way EUE device information regarding the network addresses of navigation information accessible through DOCSIS modem multicast services. Its format is defined in Table 22.

Table 22 – IP Provisioning Message Descriptor

Syntax	No. of Bits	Identifier
IP_provisioning_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
descriptor_version_number	5	uimsbf
reserved	3	bslbf
descriptor_body(){		
NIT_IP	32	uimsbf
NIT_port	16	uimsbf
BAT_IP	32	uimsbf
BAT_port	16	uimsbf
SDT_IP	32	uimsbf
SDT_port	16	uimsbf
TOT_IP	32	uimsbf
TOT_port	16	uimsbf
CAT_IP	32	uimsbf
CAT_port	16	uimsbf
}		
CRC-32	32	rpchof
}		

descriptor_tag: An 8-bit field that shall be set to **0xBE** to identify this descriptor uniquely as an IP Provisioning message descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

descriptor_version_number: A 5-bit field that shall that contains the version number of the IP_provisioning_header( ) portion of the section. It shall be incremented by one whenever a change in the information carried within the header portion occurs.

NIT_IP,
BAT_IP,
SDT_IP,
TOT_IP,
CAT_IP: These 32-bit fields provide the TCP/IP or UDP addresses for each SI data service respectively. The data shall be organized as a packed set containing the four octet bytes as aa.bb.cc.dd.

NIT_port,
BAT_port,
SDT_port,
TOT_port,
CAT_port: These 16-bit fields provide the port addresses associated with each SI data service, respectively.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in **[1]**, annex B after processing the entire descriptor.

4.5.2 DDC Message Descriptor

Table 23 – DDC Message Descriptor Syntax

Syntax	Bits	Format
DDC_message_descriptor(){		
descriptor_tag	8	uimsbf
download_descriptor_header(){		
descriptor_version	32	uimsbf
protocol_version	16	uimsbf
reserved	16	uimsbf
descriptor_message_length	32	uimsbf
descriptor_count	32	uimsbf
header_CRC32	32	uimsbf
}		
for (I=0;I<descriptor_count;I++){		
download_image_descriptor()		
}		
DDC_signature	160	uimsbf
}		

descriptor_tag: An 8-bit field that shall be set to **0xBF** to identify this descriptor uniquely as a DDC message descriptor.

descriptor_version: A unique 32-bit number identifying this particular SBS download descriptor. Its purpose is to distinguish old DDC message descriptors from new ones.

protocol_version: A 16-bit field whose function, in future, is to allow the ***DDC_message_descriptor*** structure to carry parameters that may be structured differently from those defined in the current protocol. A EUE may only process descriptors whose ***protocol_version*** matches an internally hardcoded protocol version. The version defined for this activity shall be **0x01**.

descriptor_message_length: This 32-bit field indicates the number of message bytes following the **download_descriptor_header( )**, including the ***digital_signature*** field.

descriptor_count: This 32-bit field indicated the number of image descriptors contained in the message.

header_CRC32: This 32-bit value is a CRC32 of the protocol header bytes from beginning of ***descriptor_version*** to end of the ***descriptor_count*** field. Its purpose is to verify correctness of the header. It allows validation of header integrity without having to wait for the section completion.

DDC_signature: This 160-bit field is used to authenticate the download descriptor message based upon the HMAC standard **[9]** and using the software download HMAC key. The value is calculated over the entire descriptor header and payload.

Table 24 – Download Image Descriptor Syntax

Syntax	Bits	Format
download_image_descriptor(){		
EUE_model_ID	32	uimsbf
version_info	32	uimsbf
Upgrade_mode	32	uimsbf
module_count	32	uimsbf
for (I=0;I<**module_count**;I++){		
download_module_descriptor()		bslbf
}		
}		

descriptor_length: Total length in bytes of the descriptor, including the length field.

EUE_model_ID: Model ID of the EUE that this can accept images/modules defined in this descriptor.

Current Definitions for this field are:

Table 25 – EUE Model IDs for Download Description

Value (Hex)	*Value (Decimal)*	*Model*
0x1001	4097	ADB One-Way
0x1101	4353	ADB Two-Way
0x2001	8193	QNS One-Way
0x2101	8449	QNS Two-Way

version_info: Version information. This is the version information that the EUE will use for comparison of image versions. It is in the form major (16 bits), mid (8 bits) and minor (8 bits). For example, 0x00010407 refers to version 1.4.7.

upgrade_mode: A 32-bit field describing the mode of the upgrade. The following modes are defined:

Table 26 – Upgrade Modes for Download Descriptors

Value	*Mode*
0x00	Reserved
0x01	Normal
0x02	Beta upgrade
0x03	Immediate upgrade
0x04-0xFFFFFFFF	Reserved

Normal upgrade: A normal upgrade is the standard image type. The SBS-Client would normally be informed of the availability of the upgrade via the upgrade agent. The upgrade agent based on headed and potentially some user parameters determines the timing of the upgrade.

In the case of an error recovery upgrade where some FLASH memory corruption has rendered the current code unusable, the bootrom SBS-Client would attempt to upgrade with the highest normal image available, immediately.

Beta upgrade: This mode identifies the image as a beta image. Only pre-selected EUEs, selected by the cable headend/source via the upgrade agent, are allowed to accept this type of upgrade. This mode is used in conjunction with upgrade agent message notification from the headend/source. The SBS-Client shall never take a beta image for an error recovery upgrade. Beta upgrades can only happen via upgrade agent control.

Immediate upgrade: This mode indicates to the EUE that it must immediately upgrade to a specific version. This mode is primarily used in conjunction with upgrade agent message notification from the headend/source. When the upgrade agent receives force upgrade notification from the headend/source, it invokes the SBS-Client to receive the forced upgrade of a headend/source-defined version. Only one forced upgrade version for any particular model may be in the upgrade carousel at any one time. The bootrom SBS-Client may optionally on power-up search for an applicable forced upgrade before booting from the FLASH. This is to allow forced upgrading of EUEs should there be some failure/incompatibility in the headend/source to upgrade agent communications/protocol.

module_count: This field indicates how many modules comprise this Image. A module refers to a unique object within the download module carousel.

Table 27 – Download Module Descriptor Syntax

Syntax	Bits	Format
download_module_descriptor(){		
module_version	32	uimsbf
module_id	32	uimsbf
location_info_count	32	uimsbf
for (I=0;I<location_info_count;I++){		
locationInfo_descriptor()		
}		
}		

module_version: This 32-bit version field indicates the version number of the module. It follows the same version syntax as the image **version_info** field.

module_id: This 32-bit field provides a unique identifier for the module type. Modules of a certain type would have the same **module_id** and be distinguished by version information. For example, bootable modules might have id=0x01, screen bitmaps might have module id= 0x02. For complete, monolithic images, this field shall be set to **0x00**.

location_info_count: This field indicated how many location descriptors are used to identify frequency/PID locations for this module. To increase error and noise immunity, any module may reside in multiple different carousels on different frequencies.

Table 28 – Location Info Descriptor Syntax

Syntax	Bits	Format
location_info_descriptor(){		
module_frequency	32	uimsbf
reserved	3	bslbf
module_PID	13	uimsbf
module_download_id	32	uimsbf
module_timeout	32	uimsbf
}		

module_frequency: This field indicates which frequency the EUE should tune to to receive this module.

module_PID: This field indicated the PID to demultiplex to obtain the module data.

module_download_id: This field indicates the unique download ID of the image so that it can be accessed quickly.

module_timeout: This value (0.1 Second units) indicates how long the EUE should wait on the defined frequency and PID for the Module. If the timeout expires, the EUE should try the next available location for the module, if defined. This value is should be set as a function of the module size, carousel data rate with some adjustment to allow for error recovery. For example, the carousel repetition time of a module might be 100 seconds, one could set this value to 100*10=1000, however this assumes that the client will receive all packets correctly and not need to stay on the carousel for more than one cycle. This type of setting is too strict and a broader value such as **carousel_repetition_time***6 (*10 for 0.1 sec units) would be more effective.

4.6 Module Delivery Protocol

Upgrade modules are carried in a private data stream, split into multiple ***module_delivery_sections()*** and these sections are mapped into MPEG-2 transport packet payloads. The PUSI bit of the MPEG-2 packet header being set indicates the start of a section. Subsequent packets on the same PID contain the remainder of the message. One section must complete before another may start. The maximum section size is 4096 bytes.

Table 29 details the syntax for the module delivery section protocol:

Table 29 – Module Delivery Schedule Syntax

Syntax	Bits	Format
module_delivery_section(){		
protocol_version	16	uimsbf
Reserved, value ≡ 0xCDCD	16	uimsbf
EUE_model_id	32	uimsbf
download_id	32	uimsbf
module_id	32	uimsbf
version_info (Major (16).Mid(8).Minor(8))	32	uimsbf
section_count	32	uimsbf
current_section	32	uimsbf
Image_data_size	32	uimsbf
Image_data_format	32	uimsbf
encryption_mode *(reserved)*	32	uimsbf
Header_CRC32	32	uimsbf
for (I=0;I<image_data_size;I++){		
upgrade_image_data	8	bslbf
}		
section_signature	160	uimsbf
}		

protocol_version: A 16-bit field whose function, in future, is to allow the **module_delivery_section( )** structure to carry parameters that may be structured differently from those defined in the current protocol. A EUE may only process sections whose ***protocol_version*** match an internally hardcoded protocol version.

The ***protocol_version*** shall be **0x0001** for the *Simultrans* activity

EUE_model_id: This 32-bit field identifies the EUE model that is allowed to receive this particular upgrade.

Table 30 – EUE Model IDs for Module Delivery

Value (Hex)	*Value (Decimal)*	*Model*
0x1001	4097	ADB One-Way
0x1101	4353	ADB Two-Way
0x2001	8193	QNS One-Way
0x2101	8449	QNS Two-Way

download_id: This 32-bit field defines a unique id for each upgrade image in the carousel. Once a EUE has identified the image within the carousel based on ***upgrade_mode*** and ***model_id*** it can then use the ***download_id*** to speed up recognition of EUE upgrade protocol sections.

module_id: This 32-bit field defines a unique id for each separate installable component for devices that store an image in a segmented fashion, such as a file structure. This could be used to identify device drivers, fonts, bitmaps and other component images. For complete, monolithic images, this field shall be set to **0x00000000**.

version_id: This 32-bit field identifies the version number of this upgrade image. A value of zero for this field is not allowed. The 32-bit value is sub-defined into Major, Mid and Minor version fields in the following manner:

```
UINT16 majorVersion;
UINT8  midVersion;
UINT8  minorVersion;
```

The version number can be visualized in dot notation as Major.Mid.Minor. This allows version definitions from 0.0.1 to 65535.255.255.

section_count: This 32-bit field identifies how many EUE upgrade protocol sections comprise the upgrade image.

current_section: This 32-bit field indicates which section of the upgrade image is currently being received. Use of this field allows the SBS-Client to start assembling the upgrade image starting from any section in the carousel. No attempt to apply the upgrade image shall be made until all sections have been assembled and verified. The first section shall start at zero.

image_data_size: This 32-bit field indicates the size of the upgrade image data contained within this section. The upgrade image data starts immediately after the header information, terminated by the ***header_CRC32***, ending with the last upgrade image data byte and does not include the signature field.

image_data_format: This 32-bit field identifies the actual packed format of the image data and determines how the SBS-Client will unpack the image data. The following types are defined:

Table 31 – Packed File Format Identifiers

Value	*Format*
0x00	Reserved
0x01	Reserved
0x02	Packed Multi-File Format
0x03	Reserved
0x04-0xFFFFFFFF	Reserved

Packed image formats contain one or more files or data elements packed into one image according to a pre-defined format. The image data will contain a packing header, which shall be used by the unpacking routine to recover the files correctly.

encryption_mode: This field is reserved.

header_CRC32: This 32-bit value is a CRC32 of the protocol header bytes from ***protocol_version*** to the ***header_CRC32*** field. Its purpose is to verify the validity of the header and allows correct interpretation of the header without waiting for the section to complete.

section_signature: This 160-bit field is used to authenticate and confirm reception of the section based upon the HMAC standard **[9]** and using the software download HMAC key. The signature is calculated over the entire section payload, less MPEG packet headers.

4.7 Upgrade Stream Format

The SBS protocol shall be encapsulated in MPEG-2 Transport stream packets according to User Private Stream data type. The SBS-Protocol sections shall be broken down into multiple MPEG-2 transport packets and mapped into the 184-byte payloads.

An SBS protocol section header is always mapped into a transport packet, whose Payload Unit Indicator (PUSI) is set to **1**. All other transport packets containing data for that section shall have their PUSI set to **0**. The SBS protocol section size should be an integer multiple of 184 bytes, for efficiency. The MPEG-2 transport packet containing the last data of a section may optionally pad the unused bytes of the packet to 0x00 or 0xFF. The padding shall be ignored by the SBS-Client.

SBS protocol sections have a maximum possible length of 4Gbytes, but in actual use, a length of 184Kbytes or less would be more practical. The actual section length shall be determined by the server and may be based on:

- The data rate available for upgrade carouselling
- The number of images in the carousel
- The average upgrade image size
- Hardware limitations in the receiving EUE

4.8 Upgrade Image Encryption

All transport packets received as part of the SBS protocol shall be encrypted using the ETSI-DVB-CSA standard, configured with the single fixed key pair defined by the application.

4.9 Upgrade Authentication

Authentication of upgrades is performed at three discrete levels:

- Authentication of headend/source messages.
- Authentication of SBS download descriptors.
- Authentication of individual module images on the module image carousel.

SBS download descriptors (DDC) and module images are both signed using the HMAC standard **[9]** using a dedicated download HMAC key,defined by the application. A corresponding signature checking function shall reside in the EUE, allowing authentication of signatures for each SBS download descriptor and module image section.

The signature is calculated over the entire section payload, less MPEG packet headers.

The headend/source message is passed to the EUE via an encoded EMM. This EMM has both encryption and authentication. Its format is described in detail in section 0 of this specification.

4.10 Upgrade Agent to SBS–Client Interface

The upgrade agent is an independent task running at the application level of the EUE software. The upgrade agent performs the following duties:

1. Receives authenticated upgrade notification messages from the headend/source.
2. Processes upgrade notification messages. The possible notification messages are:

- Normal upgrade
- Force upgrade
- Schedule_upgrade
- Beta upgrade

3. Handle the timing of invoking the SBS-Client to receive the upgrade image.

The upgrade agent shall provide, in conjunction with the SBS-Client, the capability to provide a managed and addressable upgrade scheme. As a minimum, the SBS-Client shall provide enough functionality to perform EUE upgrades without the presence of the upgrade agent. The SBS upgrade protocol provides this flexibility by supporting basic upgrade capability along with a special agent interface facilitating a more flexible, yet complex, upgrade mechanism.

The upgrade agent passes commands known as ***push_command_set*** requests to the SBS-Client, which are run only at boot time from the bootrom via a message it stores in nonvolatile memory. The SBS-Client searches for the presence of the ***push_command_set*** message at boot time, including an immediate reboot invoked by the upgrade agent. If this message exists, then the SBS-Client can determine that the upgrade agent has a pending upgrade request and the specific upgrade instructions are contained within the message.

The ***push_command_set*** message is a binary record that uses the following protocol:

Table 32 – Push Command Set Message Syntax

Syntax	Bits	Format
push_command_set(){		
protocol_version	16	uimsbf
reserved	15	uimsbf
beta_flag	1	uimsbf
EUE_model_id	32	uimsbf
version_info (Major (16).Mid(8).Minor(8))	32	uimsbf
CRC32	32	uimsbf
}		

protocol_version*:* A 16-bit field whose function in future is to allow the ***push_command_set*** record to carry parameters that may be structured differently from those defined in the current protocol. A EUE may only process sections whose ***protocol_version*** matches an internally hardcoded protocol version.

The ***protocol_version*** value for the *Simultrans* activity shall be 0x01.

beta_flag A 1-bit flag that when set to one, signifies that the EUE should take an image marked as type **BETA**, if available. If the bit is set to zero (default), the EUE should take type **FORCED** or **NORMAL** releases, as appropriate.

EUE_model_id*:* This 32-bit field identifies the EUE model that is allowed to receive this particular upgrade.

Table 33 – EUE Model IDs for Push Command Set

Value (Hex)	*Value (Decimal)*	*Model*
0x1001	4097	ADB One-Way
0x1101	4353	ADB Two-Way
0x2001	8193	QNS One-Way
0x2101	8449	QNS Two-Way

***version_info**:* This 32-bit field identifies the version number of the intended upgrade image. A value of zero for this field is allowed and shall be interpreted as "take highest version". The 32-bit value is sub-defined into Major, Mid and Minor version fields in the following manner:

```
UINT16 majorVersion;
UINT8  midVersion;
UINT8  minorVersion;
```

The version number can be visualized in dot notation as Major.Mid.Minor. This allows version definitions from 0.0.0 to 65535.255.255.

CRC32: This 32-bit value is a CRC32 of the data structure from ***protocol_version*** upto but not including the ***CRC32*** field.

4.11 EUE Version Information Storage

This section describes the method used to store version control information within the EUE. There shall be an overall release version identifier, which is obtained from the image header ***version_info*** field upon software upgrade. This version information indicates the last upgrade that was applied to the EUE. The version of the downloaded image may have consisted of one or more component modules, with each of the component modules having its own distinct version and module identifier code. It is expected that module ids shall be pre-allocated and assigned based on the content of the module. The EUE will keep a record of the image version, the module version and ids for each of the modules that comprised that particular image.

If a new upgrade is desired, then a new image descriptor should be placed in the SBS download descriptor carousel with a version number that is greater than the previous highest version number. The new descriptor may refer to a completely new set of modules, in which case all the new modules must be placed on the module carousel or it the new descriptor may be comprised of modules from the previous version, with one or more updated modules. The upgrade client shall be able to determine which modules are up-to-date and upgrade only those modules described by the new image descriptor, which are different from those installed previously. This process allows incremental upgrades and provides a simpler method of tracking installed versions.

The version information is stored in the EUE as a binary record, residing in nonvolatile memory. Its contents are defined as:

Table 34 – Upgrade Record Format

Syntax	Bits	Format
upgrade_record(){		
record_count	32	uimsbf
image_version	32	uimsbf
header_CRC32	32	uimsbf
for (I=0;I<record_count;I++)		
{		
upgrade_mode	32	uimsbf
module_id	32	uimsbf
module_version	32	uimsbf
upgrade_path (reserved)	32	uimsbf
}		
Upgrade_CRC32	32	uimsbf
}		

record_count: This indicates how many records are in the file.

image_version: The image version of last upgrade.

header_CRC32: CRC32 of all header fields.

upgrade_mode: The mode that this particular module was upgraded by.

module_id: The module ID of this module.

module_version: The version of this particular module.

upgrade_path: Reserved

upgrade_CRC32: The CRC32 of all record data after the ***header_CRC32***.

4.12 Boot-time Software Validity Checking

If any upgrade module file has been tagged as a CRITICAL_COMPONENT_FILE (section 0), then the upgrade client shall keep a record of those files and their associated CRC32 values so that a validity check can be performed on those critical files at each boot-time. The CRC/module information is stored in the EUE as a binary record in nonvolatile memory. Its contents are defined as:

Table 35 – CRC Record Format

Syntax	Bits	Format
crc_record(){		
record_count	32	uimsbf
image_version (reserved)	32	uimsbf
header_CRC32	32	uimsbf
for (i=0;i<**record_count**;i++)		
{		
module_CRC32	32	uimsbf
module_name_length	8	uimsbf
for (j=0;j<**module_name_length**;j++)	32	
module_name	8	bslbf
module_version	32	uimsbf
module_ID	32	uimsbf
}		
record_CRC32	32	uimsbf
}		

record_count: This field indicates how many entries are in the structure.

image_version: reserved

header_CRC32: CRC32 of all header fields.

module_CRC32: The CRC32 of the module in this record

module_name: The module name associated with this record

module_version: The version of this particular module.

module_id: The module ID of this module.

record_CRC32: The CRC32 of all record data after the ***header_CRC32***.

4.13 Download Image Preparation

This section provides details of the process for packagingelivery of EUE code modules/files to the headend/source upgrade server, describing how device image files and version control information files are

packed into one upgrade image file. The image file is then encapsulated in the SBS-Upgrade client module protocol for carouselling in the in-band data path. It is intended that the packing format be as simple as possible to allow for easy and quick retrieval of the contents. This code shall be implemented in the SBS-Client, residing in inerasable bootrom. Because of the static nature of the bootrom device, it is desirable that this process be simple and flexible thereby avoiding the need for multiple bootrom versions or EUE rework. The upgrade server shall encapsulate the packed image according to the methods described later in this document and the EUE shall retrieve the packed image and recreate the original files according to process described in this section.

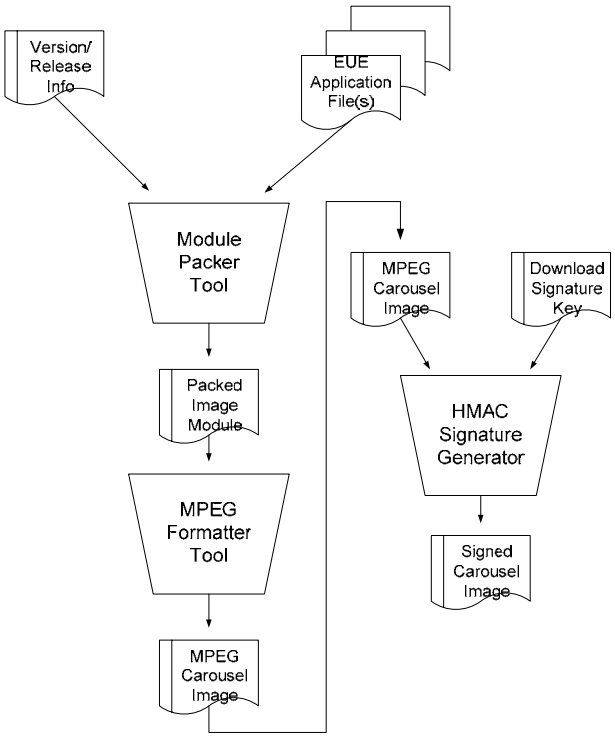


Figure 11 – Module Creation Process

The configuration file shall be a text based script file interpreted by the packing utility.

The preparation process follows these steps:

1. **Module Packing** — Converts an image file to a packed binary format and adds an identifying header structure using the **modulePack** utility.
2. **MPEG Formatting** — Converts the modified packed file to MPEG-2 format using the **module2mpeg** utility.
3. **Module Signing** — Calculates the security signature for the packed module sections using the **moduleSign** utility.

The implementation of all these utilities may be combined into a singular operational process through scripts and batch files. Following module and DDC preparation, the respective content is loaded on the carousel server for play out. DVB encryption of the download stream is performed in real-time using the remultiplexer receiving the MPEG-2 stream or by the Passage processor.

4.13.1 Module Packing Process

The headend/source packing of the upgrade image shall be performed by a DOS, Windows™ or Linux based utility that takes as input a configuration file. The configuration file shall contain information describing the files to be packed into the image module, version information and image specific parameters used by the upgrade server to control the upgrade process.

The packing utility will combine all the files into one flat image module file and pre-pend a header to the start of the file, which shall contain information to recover the original files correctly.

4.13.1.1 Packed File Format

The packed file shall be a binary file with the following format:

```
struct packedHeader Packed_File_Header;
char imageTitle[titleLength];
char imageDescription[descriptionLength];
for (i=0;i<packed_file_count;i++)
{
        char    filename[256];
        UINT32  fileAttribute;
        UINT32  fileFlags;
        UINT32  fileCRC32;
        UINT32  fileSize;
        char    FileData[sizeof(Data_File)];
}
```

The file begins with a fixed header containing information needed by the upgrade server to construct the image data carousel correctly.

Packed_File_Header: The Packed_File_Header structure shall have the following fields:

```
typedef struct packedHeader
{
    UINT32   targetModel;
    UINT32   moduleId;
    UINT32   versionInfo;
    UINT32   encryptionMode;
    UINT16   titleLength;
    UINT16   descriptionLength;
    UINT32   packedFileCount;

} PACKED_FILE_HEADER;
```

targetModel: This 32-bit field identifies the model type of EUE that can accept this image.

moduleID: This 32-bit field identifies this module ID of this image. Different module ID's would be used for different components of the system, e.g. bitmaps, fonts, device drivers, etc.

versionInfo: This field is used to represent the version number of the upgrade image.

encryptionMode**:* This field represents the ***encryption_mode field of the SBS-Client Protocol and is used to specify if any encryption should be applied to the image file before encapsulating in the SBS-Client protocol. Encryption mode 0 represents unencrypted images. For the *Simultrans* activity, all images shall be type 0.

titleLength,
***descriptionLength**:* These fields specify the length in bytes of the null terminated ASCII strings following the header which identify the title and description of the upgrade image.

***packedFileCount**:* This field identifies the number of files packed into the upgrade image.

imageTitle,
imageDescription: The ***imageTitle*** and ***imageDescription*** fields are null terminated ASCII strings.

filename: A null terminated ASCII string of max 256 bytes which holds the filename associated with the preceding ***fileBytes***.

fileAttribute: The attribute field can be used to specify how a particular file should be supplied.

The following file attributes are specified:

Value	*Meaning*
0x00	Reserved
0x01	Reserved
0x02	Component File
0x03	Erase File

A *Component File* attribute describes the preceding file as a standard component and should not be treated in any special way by the upgrade client. This could be a bootable file but the client shall not modify the boot line.

An *Erase File* attribute should be attached to a filename and a zero length file. Its purpose is to allow the headend/source to delete specific files residing on the EUE. The upgrade client will delete files with the corresponding filename off the local file system on the EUE.

fileFlags: MSB indicates CRC field contains valid data, signifying a critical component file if the ***fileAttribute*** value is 0x02.

fileSize: This 32-bit unsigned integer defines the size, in bytes, of the file.

fileCRC32: If the fileFlags indicate the CRC is valid, then a 32-bit CRC32 is calculated over the entire file payload.

fileBytes: The actual bytes representing the file.

4.13.1.2 Creation of Packed Module Files

A packed file is generated by collecting the required component files for an upgrade image into a hard disk directory. A configuration script is then created, using a text editor, describing the filenames and providing version and description information that will determine how the Upgrade Image is created and ultimately applied at the EUE.

A configuration script has the following format:

IMAGE_VERSION = major.mid.minor
MODULE_ID = xxxxx
IMAGE_TARGET = model_id
ENCRYPTION_MODE = 0..255
IMAGE_TITLE = ”Some title text”
IMAGE_DESCRIPTION = ”Some description text”
ERASE_FILE = userPref.dat
COMPONENT_FILE = drivers/ramDriver.sys,drivers/serialDrv.sys,fonts/sanSerrif.fnt
COMPONENT_FILE = Icons/Sony.ico,icons/xxxx.ico
CRITICAL_COMPONENT_FILE = browser.exe

The parameters have the following rules:

MODULE_ID:	specifies a decimal ID for the module.
IMAGE_TARGET:	model ID must be specified in decimal (32 bits).
IMAGE_VERSION:	major, mid and minor values are in decimal and separated with a period, max 65535.255.255, min 0.0.1.
ENCRYPTION_MODE:	must be specified in decimal, defined as ‘0’.
IMAGE_TITLE, **IMAGE_DESCRIPTION**:	must be enclosed in double quotes.
COMPONENT_FILE, **CRITICAL_COMPONENT_FILE**, **ERASE_FILE**:	must be DOS compatible 8.3 filenames. May be multiple files separated by comma.

The **ERASE_FILE** field should be used with care. Its purpose is to direct the upgrade client to delete certain files off the EUE file system. The EUE upgrade client will process the files specified in the _FILE fields in the order they are entered into the configuration script. The following example shows a combination that would result in the loss of the boot file:

COMPONENT_FILE = vxWorks.st
ERASE_FILE = vxWorks.st

This occurs because the EUE Upgrade client would see the erase command after it had copied the boot file to the local file system thereby deleting it.

Multiple files may be specified over several **ERASE_FILE** or **COMPONENT_FILE** fields, enhancing readability of the configuration script.

Files defined as **CRITICAL_COMPONENT_FILES** will have CRC32 checking enabled. This causes the upgrade client to check the CRC after writing to FLASH and to check the CRC of that file every time the EUE boots up.

Scripts should be saved with the “.scr” , SCRipt, file extension.

Once a script has been created then the files can be packed by running the **modulePack**.exe utility, which uses the following syntax:

modulePack –i <input config scriptfile> –o <output image file>

For example:

modulePack –i v101.scr –o v101.pak

For consistency, the ".pak", extension should always be used for the output packed image files from the modulePack utility.

4.13.2 Conversion of Packed Files to MPEG-2 Format

Packed image module files must be converted into MPEG-2 formatted files for playout by the headend/source MPEG-2 packetizer device. The module2mpeg.exe utility is used to perform this conversion. It reads a packed image module file and segments the file into 4096 byte "MPEG-like" private sections of data, packaged in 188 byte MPEG-2 transport packets. The utility creates appropriate section header data and inserts MPEG-2 packet headers with correct continuity count values, PUSI bits and PID (if specified). **module2mpeg.exe** uses the following syntax:

module2mpeg -i <input filename> -o <output filename> -d <download id> -p <output pid>
-i: Input module filename
-o: Output MPEG-2 filename
-d: Download ID to use (decimal only)
-p PID to assign to created MPEG packets (decimal only, optional)

For example:

module2mpeg –i bigimage.pak –o bigimage.mpg –d 11062 –p 256

For consistency, the ".mpg", extension should always be used for the output mpeg files from the **module2mpeg** utility.

Each image file is becomes part of a module within an upgrade carousel in the MPEG-2 transport stream. To identify each image uniquely, all its component modules must share a ***download_id*** unique to a particular image. No two images shall have the same ***download_id***.

4.13.3 Module Signing

Once download data has been formatted into an MPEG-2 compliant format, it must have a cryptographic signature applied to allow EUE devices receiving the data to confirm both the integrity and validity of the data. The cryptographic signature allows detection of content corrupted at any stage in transmission as well as detection of tampering or substitution of unauthorized content, allowing the EUE devices to reject the noncompliant material.

The signature consists of a 160 bit string (message digest) that is calculated using a public algorithm **[9]**, but employs a secret key value unique to the specific application activity. The secret key value is provided by the application. A data file containing the key is read by the signing utility along with a packetized image module file and the HMAC signature calculated across the entire section structure, MPEG-2 header bytes excluded. Each 4096 byte section in a download image module has a separately calculated signature. **moduleSign.exe** uses the following syntax:

moduleSign <Key file name> <MPEG image module filename>

- Key filename — The name of the file containing the secret key to use for signing the module sections.
- MPEG image module filename — The name of the file containing the MPEG-2 packetized download image module data created by the module2mpeg utility.

For example:

moduleSign downloadkey.hex bigimagefile.mpg

4.14 Provisioning Message Preparation

This section provides details regarding the process for packagingelivery of EUE provisioning messages and code download announcement messages to the headend/source delivery system, describing how SI provisioning, IP provisioning and signed DDC messages are packed into one provisioning message section structure. The provisioning message is then encapsulated in MPEG-2 transport layer protocol for carouselling in the in-band data path. It should be that the packing format be as simple as possible to allow for easy and quick retrieval of the contents and interpretation/use by the receiving EUE.

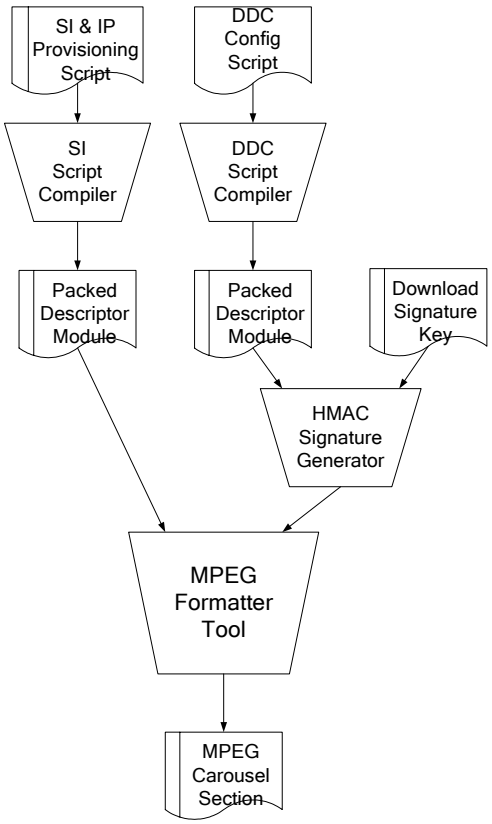


Figure12 – Provisioning Message Creation Process

The provisioning and configuration files shall be text based script files interpreted by the respective compiler utilities.

The preparation process follows these steps:

1. **DDC Message Creation** Converts a text script to a formatted DDC descriptor using the **ddcMake** utility.
2. **DDC Signing** Calculates the security signature for the packed module sections using the **ddcSign** utility.
3. **SI Provisioning Descriptor Creation** Converts a text script to a formatted provisioning message using the **pmessMake** utility.

4. **Provisioning Message MPEG Formatting** — Concatenates the modified packed DDC descriptor and provisioning messages and then converts the composite to MPEG transport format using the **pmess2mpeg** utility.

The implementation of all these utilities may be combined into a singular operational process through scripts and batch files. Following message preparation, the mpeg section is loaded on the carousel server for play out.

4.14.1 DDC Message Creation

The SBS upgrade client relies on the use of SBS download descriptors to identify the location of download images. The descriptors are compiled from a supplied script text file with the following utility.

ddcMake -i filename -o filename
-i: specifies the input script filename
-o: specifies the output SBS download descriptor mpeg formatted filename

The input file is a script file describing the images available and the locations of the associated modules. An image can comprise of one or more modules. Each module has a location descriptor defining where it can be found. Each module may have several location descriptors to allow for error resilience.

For example:

ddcMake ddcData.scr ddcDataFile.pak

4.14.1.1 DDC Message Descriptor Compiler Script Syntax

All DDC message scripts must be encapsulated in paired, nested blocks establishing the context for the enclosed syntax. The outermost syntax block is delimited the pair:

DOWNLOAD_DESCRIPTOR_START
…
…
DOWNLOAD_DESCRIPTOR_END

There may only be one **DOWNLOAD_DESCR**IPTOR text pair per script.

The second nested block is delineated by the syntax pair:

IMAGE_DESCRIPTOR_START
…
…
IMAGE_DESCRIPTOR_END

The contents within the paired **IMAGE_DESCR**IPTOR statements are contextually associated with a single download image for a specific EUE device. There may be multiple IMAGE_DECRIPTOR blocks contained within a download script file, but the blocks must be sequential and non-overlapping in scope.

Located within an **IMAGE_DESCRIPTOR** block are characteristics statements that establish the characteristics for the particular download image context in which the statements occur. All characteristics statements are followed by an assignment operator (=) and then the value to be used for the particular statement. The order of statements within the block is unimportant. The possible characteristics statements for an **IMAGE_DESCRIPTOR** block are:

IMAGE_TARGET	Specifies the EUE manufacturer/model code (decimal) applicable to the image.
IMAGE_VERSION	Specifies the version for the overall image. The format is (decimal) <major>.<mid>.<minor>
UPGRADE_MODE	Specifies the priority (mode) associated with the image. The possible values are:

NORMAL	Image is downloaded at next reboot or after EMM directive.
SCHEDULE	Image is downloaded at the timeate specified in an EMM directive.
BETA	Image is only available to selected EUE participating in a limited distribution of the image.
FORCE	Highest priority. All EUE of matching make/model must download this image.

The third level nested block is delineated by the syntax pair:

MODULE_DESCRIPTOR_START
…
…
MODULE_DESCRIPTOR_END

The contents within the paired **MODULE_DESCRIPTOR** statements are contextually associated with a single module associated with a specific download image for a specific EUE device. There may be multiple **MODULE_DESCRIPTOR** blocks contained within any particular **IMAGE_DESCRIPTOR** block of a download script file, but the **MODULE_DESCRIPTOR** blocks must be sequential and non-overlapping in scope.

Located within a **MODULE_DESCRIPTOR** block are characteristics statements that establish the characteristics for the particular download image module context in which the statements occur. All characteristics statements are followed by an assignment operator (=) and then the value to be used for the particular statement. The order of statements within the block is unimportant. The possible characteristics statements for a **MODULE_DESCRIPTOR** block are:

MODULE_ID	Specifies.
MODULE_VERSION	Specifies the version for the overall image. The format is (decimal) <major>.<mid>.<minor>

The fourth and final level nested block is delineated by the syntax pair:

LOCATION_INFO_START
…
…
LOCATION_INFO_END

The contents within the paired **LOCATION_INFO** statements are contextually associated with a single module associated with a specific download image for a specific EUE device. There may be multiple **LOCATION_INFO** blocks contained within any particular **MODULE_DESCRIPTOR** block of a download script file, but the **LOCATION_INFO** blocks must be sequential and non-overlapping in scope.

Located within a **LOCATION_INFO** block are characteristics statements that establish the characteristics for the particular download image module context in which the statements occur. All characteristics statements are followed by an assignment operator (=) and then the value to be used for the particular statement. The order of statements within the block is unimportant. The possible characteristics statements for a **LOCATION_INFO** block are:

MODULE_FREQUENCY	Specifies the transport frequency (decimal, hertz) on which the module is located.
MODULE_DOWNLOAD_ID	Specifies a unique identifier for a particular download as a shortcut for finding associated modules.
MODULE_TIMEOUT	Specifies the time (decimal, seconds) to allow for the successful download of the module.
MODULE_PID	Specifies the transport PID (decimal) on which the module is located.

The script file format should be self-explanatory and the following examples should demonstrate its use.

```
DOWNLOAD_DESCRIPTOR_START

#target image #1, this image comprises 3 modules
    IMAGE_DESCRIPTOR_START
        #destined for EUE ID 4097 (ADB 1-way Device)
        IMAGE_TARGET=4097
        IMAGE_VERSION=1.0.4
        UPGRADE_MODE=NORMAL
    #module 1
        MODULE_DESCRIPTOR_START
            MODULE_ID=001
            MODULE_VERSION=0.0.1

            LOCATION_INFO_START
                MODULE_FREQUENCY=500000000
                MODULE_DOWNLOAD_ID=1234
                MODULE_TIMEOUT=0
                MODULE_PID=1234
            LOCATION_INFO_END

            LOCATION_INFO_START
                MODULE_FREQUENCY=500000000
                MODULE_PID=7937
                MODULE_DOWNLOAD_ID=1234
                MODULE_TIMEOUT=0
            LOCATION_INFO_END
        MODULE_DESCRIPTOR_END

    # module 2
        MODULE_DESCRIPTOR_START
            MODULE_ID=002
            MODULE_VERSION=0.0.1

            LOCATION_INFO_START
                MODULE_FREQUENCY=500000000
                MODULE_PID=7938
                MODULE_DOWNLOAD_ID=1235
                MODULE_TIMEOUT=0
            LOCATION_INFO_END
        MODULE_DESCRIPTOR_END
```

```
        # module 3
                MODULE_DESCRIPTOR_START
                        MODULE_ID=0003
                        MODULE_VERSION=0.0.1
                        LOCATION_INFO_START
                                MODULE_FREQUENCY=510000000
                                MODULE_PID=7939
                                MODULE_DOWNLOAD_ID=1236
                                MODULE_TIMEOUT=0
                        LOCATION_INFO_END
                MODULE_DESCRIPTOR_END
        IMAGE_DESCRIPTOR_END


    # target image #2, comprised of one module only for QNS 2-way device
        IMAGE_DESCRIPTOR_START
                IMAGE_TARGET=8449
                IMAGE_VERSION=1.0.5
                UPGRADE_MODE=FORCE

    # module 1
                MODULE_DESCRIPTOR_START
                        MODULE_ID=001
                        MODULE_VERSION=0.0.1
                        LOCATION_INFO_START
                                MODULE_FREQUENCY=500000000
                                MODULE_PID=7937
                                MODULE_DOWNLOAD_ID=1234
                                MODULE_TIMEOUT=0
                        LOCATION_INFO_END
                MODULE_DESCRIPTOR_END
        IMAGE_DESCRIPTOR_END
DOWNLOAD_DESCRIPTOR_END
```

4.14.2 DDC Message Signing

Once DDC message data has been formatted into a packed record structure, it must have a cryptographic signature applied to allow EUE devices receiving the data to confirm both the integrity and validity of the data. The cryptographic signature allows detection of content corrupted at any stage in transmission as well as detection of tampering or substitution of unauthorized content, allowing the EUE devices to reject the noncompliant material.

The signature consists of a 160 bit string (message digest) that is calculated using a public algorithm **[9]**, but employing a secret unique key value defined by the application. A data file containing the key is read by the signing utility along with a packetized image module file and the HMAC signature calculated across the entire section structure **ddcSign**.exe uses the following syntax:

ddcSign <Key file name> <Packed DDC message filename>

- Key filename — The name of the file containing the secret key to use for signing the module sections.
- Packed DDC message filename — The name of the file containing the packed ddc message data created by the **ddcMake** utility.

For example:

ddcSign downloadkey.hex ddcDataFile.pak

4.14.3 Provisioning Message Descriptor Creation

The EUE device relies upon the use of a provisioning message containing the PID or TCP/IP address assignments for the various SI services necessary for proper navigation of the cable network. The descriptors are compiled from an input text script file with the following utility.

pmessMake -i filename -o filename
-i: specifies the input script filename
-o: specifies the output provisioning message packed format filename

The input file is a script file describing the system resources available and their locations.

For example:

pmessMake pmessData.scr pmessData.pak

4.14.3.1 Provisioning Message Descriptor Compiler Script Syntax

All provisioning message scripts must be encapsulated in paired, nested blocks establishing the context for the enclosed syntax. The outermost syntax block is delimited the pair:

PROVISIONING_MESSAGE_START
…
…
PROVISIONING_MESSAGE _END

There may only be one **PROVISIONING_MESSAGE** text pair per script.

The second nested block is delineated by the syntax pair:

SI_HEADER_START
…
…
SI_HEADER_END

The contents within the paired **SI_HEADER** statements are contextually associated with all EUE devices. There may be only one **SI_HEADER** block contained within a provisioning message script file.

Located within an **SI_HEADER** block are characteristics statements that establish the characteristics for the provisioning message header. All characteristics statements are followed by an assignment operator (**=**) and then the value to be used for the particular statement. The order of statements within the block is unimportant. The possible characteristics statements for an **SI_HEADER** block are:

NIT_PID	Specifies the transport PID associated with the network information table (NIT).
BAT_PID	Specifies the transport PID associated with the bouquet association table (BAT).
SDT_PID	Specifies the transport PID associated with the service description table (SDT).
TOT_PID	Specifies the transport PID associated with the time offset table (TOT).

CAT_PID Specifies the transport PID associated with the conditional access table (CAT).

SI_HEADER_VERSION Specifies the version number to be associated with the section header.

The third nested block is delineated by the syntax pair:

IP_DESCRIPTOR_START
…
…
IP_DESCRIPTOR_END

The contents within the paired **IP_DESCRIPTOR** statements are contextually associated with all EUE devices. There may be only one **IP_DESCRIPTOR** block contained within a provisioning message script file. Located within an **IP_DESCRIPTOR** block are characteristics statements that establish the characteristics for the provisioning message header. All characteristics statements are followed by an assignment operator (=) and then the value to be used for the particular statement. The order of statements within the block is unimportant. The possible characteristics statements for an **IP_DESCRIPTOR** block are:

NIT_IP Specifies the TCP/IP address associated with the network information table (NIT).

NIT_PORT Specifies the TCP/IP port associated with the network information table (NIT).

BAT_ IP Specifies the TCP/IP address associated with the bouquet association table (BAT).

BAT_PORT Specifies the TCP/IP port associated with the bouquet association table (BAT).

SDT_IP Specifies the TCP/IP address associated with the service description table (SDT).

SDT_PORT Specifies the TCP/IP port associated with the service description table (SDT).

TOT_IP Specifies the TCP/IP address associated with the time offset table (TOT).

TOT_PORT Specifies the TCP/IP port associated with the time offset table (TOT).

CAT_IP Specifies the TCP/IP address associated with the conditional access table (CAT).

CAT_PORT Specifies the TCP/IP port associated with the conditional access table (CAT).

DESCRIPTOR_VERSION Specifies the version number to be associated with the section header.

The script file format should be self-explanatory and the following example should demonstrate its use.

```
PROVISIONING_MESSAGE_START

  SI_HEADER_START
    NIT_PID = 401
    BAT_PID = 402
    SDT_PID = 403
    TOT_PID = 404
    CAT_PID = 405
    SI_HEADER_VERSION = 255
  SI_HEADER_END
```

IP_DESCRIPTOR_START
NIT_IP = 101.102.103.201
NIT_PORT = 511
BAT_IP = 101.102.103.202
BAT_PORT = 512
SDT_IP = 101.102.103.203
SDT_PORT = 513
TOT_IP = 101.102.103.204
TOT_PORT = 514
CAT_IP = 101.102.103.205
CAT_PORT = 515
DESCRIPTOR_VERSION = 254
IP_DESCRIPTOR_END

PROVISION_MESSAGE_VERSION = 253

PROVISION_MESSAGE_END

4.14.4 Conversion of Provisioning Message to MPEG-2 Format

The SBS upgrade client relies on the use of SBS download descriptors to identify the location of download images. The descriptors may be created as an MPEG formatted file with the following utility. The descriptor file is loaded into the headend/source MPEG-2 packetizer device and played out continuously on a unique PID.

pmess2mpeg -id filename -ip filename -o filename -p ### -v ###
-id: specifies the input packed & signed DDC filename
-ip: specifies the input packed provisioning message descriptor filename
-o: specifies the output provisioning message packed format filename
-p: specifies the MPEG PID to assign to the transport packets (decimal)
-v: specifies the assigned provisioning message version number (decimal)

Informative example:

pmess2mpeg –id ddcData.pak –ip pmessData.pak –o pmess.mpg –v 11062 –p 401

For consistency, the ".mpg", extension shall always be used for the output mpeg files from the **pmess2mpeg** utility.

5 EUE Fingerprint

This section describes a robust method to self-detect the unauthorized relocation of a one-way digital communication appliance. The method presented has high resolution, yet requires no additional hardware to be added to the product in which it is implemented. Implementation of the concept uses resources already present in all digital cable television EUE. A system for automated management is presented wherein subscribers could self-activate attached devices without need for further operator intervention.

5.1 Fingerprint Process

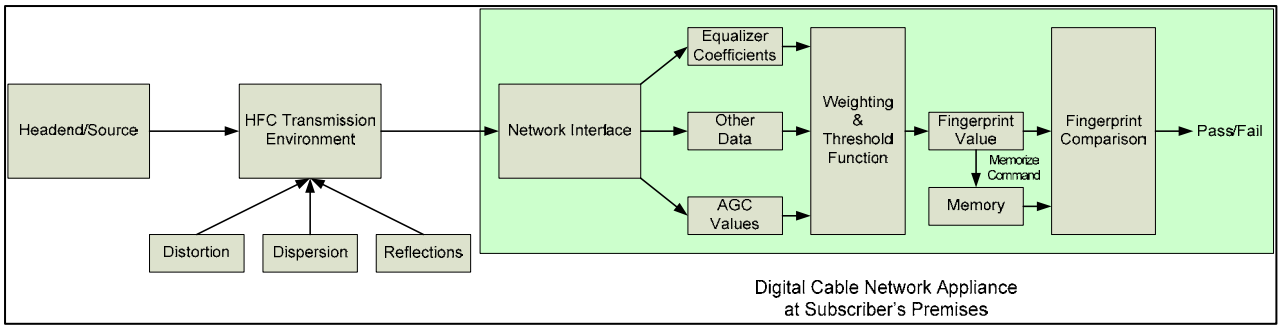


Figure 13 – RF Fingerprint System

The ability to detect changes in location of a one-way digital cable receiving device is based in large part upon the adaptive equalizer. The equalizer acts as a matched filter to the communications channel. As a result, the values contained within the equalizer's coefficients can be mathematically manipulated to show the transfer function of the communications channel that influences signals passing through it. Stated differently, the values of the coefficients, taken as a set, represent at a specific point in time the sum total knowledge of all mismatches, reflections, phase variations, gain variations, echoes and other perturbations of the transmission media upon the transmitted signal. The fact that the demodulator used in digital content delivery systems, such as digital cable television, is able to achieve and maintain signal lock in a varying signal environment validates that the state of the equalizer accurately reflects the knowledge of the plant's effect upon the system so it can negate those effects. The tolerance to a suboptimal equalizer configuration is low, given the small vector error radii for either QAM-64 or QAM-256 formats used in digital cable. The vector error radius is the composite of effects due to both amplitude and phase errors upon a received signal.

Since the filter coefficient set is directly representative of the transmission environment, it responds dynamically to any changes in that environment. The low order feedback taps are most affected by high frequency trends, such as impedance variations at the connection or connector on the back of the appliance, reflections within the cable from the house splitter(s), etc. The middle taps are more predominantly affected by variations in the characteristics of the cabling to the tap and distribution amplifier, while the highest order taps are sensitive to channel tilt, dispersion, etc. This data, when combined with the AGC information, which indicates total gain required for a constant signal level input, provides the basis for a very characteristic fingerprint of the environment where a specific cable appliance is installed.

If a complex equalizer coefficient is represented by a±jb, then $\mathbf{H}_1$, the complex matrix of all equalizer coefficients representing the state of the system at one point in time can be represented by:

$$\mathbf{H}_1 = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_n & b_n \end{pmatrix}$$

Likewise, if the gain value of one of the multiple nested AGC loops is represented by k, then H2, the matrix of all AGC coefficients representing the state of the system at one point in time can be represented by:

$$\mathbf{H}_2 = \begin{pmatrix} k_0 \\ k_1 \\ k_2 \end{pmatrix}$$

If one were to capture the equalizer tap coefficients and AGC data from a digital cable EUE, applying a thresholding function based upon the expected statistical variance and then an algorithm to allow the weighted summation of the coefficients to be reduced to a scalar, a unary value representing the unique "fingerprint" of the environment of the device could be expressed. The threshold and weighting functions could be made unique to a particular operator and are secret to reduce the likelihood of tampering.

The algorithm for these operations then looks like: Fingerprint(t) = Y($\mathbf{H}_1$,$\mathbf{H}_2$)

This fingerprint value is gathered, evaluated and stored in the EUE memory upon receipt of a command message, such as an EMM from the cable operator. The stored value should be secured through encryption and signed to detect tampering.

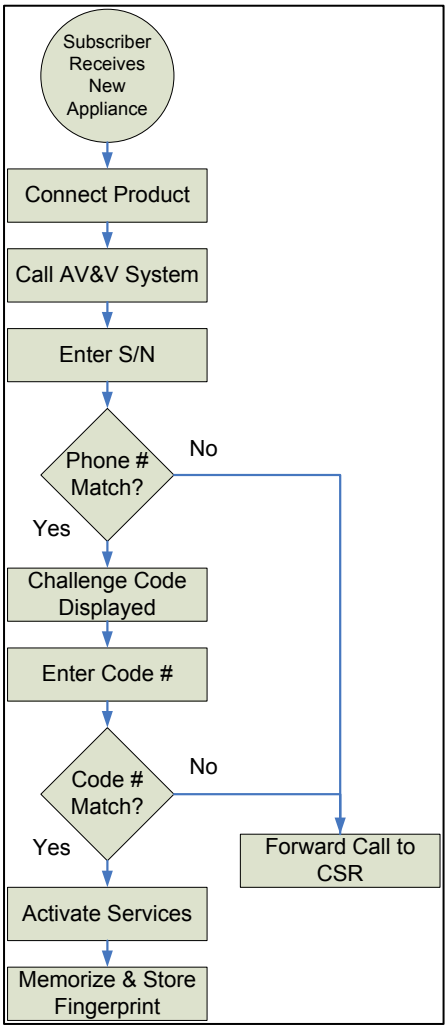


Figure 14 – EUE Challenge/Activation Process

The proposed implementation of RF fingerprinting is as follows:

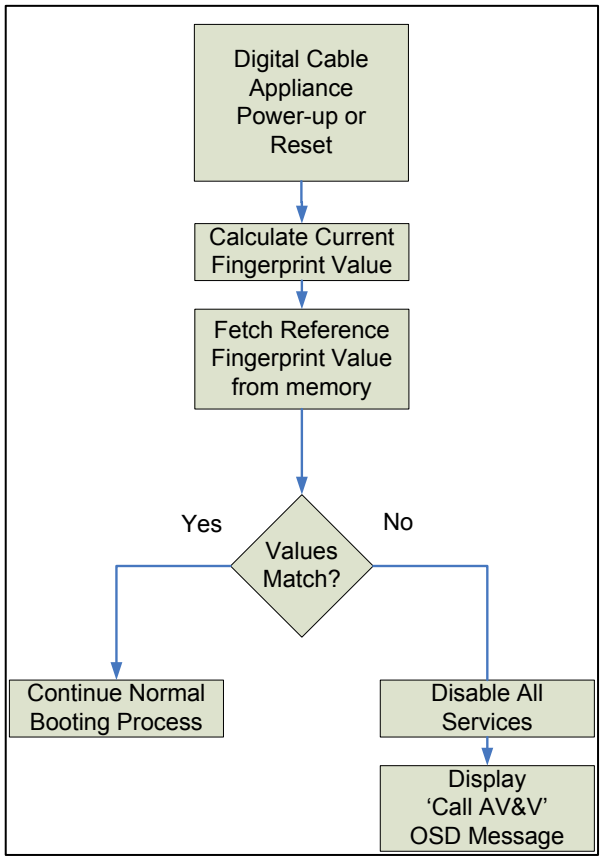


Figure 15 – Digital Cable Appliance Location Audit Process at Reboot

1. A subscriber receives a fingerprint capable EUE from the cable operator. The product contains labeling indicating "Call XXX-XXX-XXXX from your home phone after connecting the device to both the cable network and television for activation". This is identical in concept to the process now followed for activation of home satellite television receivers and credit or ATM cards issued through the mail by major financial institutions.

2. The subscriber follows the instructions and calls the number on the EUE label after installation, as indicated. An automated validation and activation (AV&V) system at the cable operator receives the call and prompts the subscriber to enter the serial number of the device using the keypad on the telephone and to press the "#" key upon completion.

3. Upon receiving the "#" key input, the AV&V system confirms the validity of the entered appliance serial number. The system then looks through its subscriber database and finds the record indicating the subscriber issued the device having the entered serial number. It then reads the subscriber's home phone number from the database record. Using ANI, a non-maskable form of caller identification used for calls to toll-free telephone numbers (and 911 calls), the AV&V system then confirms a match between the number of record and if unsuccessful, refers the call to a customer service agent. This step validates that an authorized subscriber is attempting to activate the appliance issued to them by the cable operator.

4. If the ANI and phone number of record match, the AV&V system then sends an EMM containing a fingerprint descriptor message to the EUE having the serial number the subscriber entered by phone. This EMM commands the device to display, on the subscriber's television screen, a four-digit challenge number sequence, contained within the EMM message in the ***challenge_string*** field and generated at random by the AV&V system. The AV&V system and OSD then instructs the subscriber to enter into the telephone, the number sequence displayed on the screen using the keypad of the telephone and to press the "#" key upon completion.

5. Upon receiving the "#" key input, the AV&V system confirms the validity of the entered challenge number and if unsuccessful, refers the call to a customer service agent. This step validates that the authorized subscriber is attempting to activate the appliance issued to them by the cable operator at the home of record.

6. If the challenge sequence is successful, the AV&V system sends another EMM to the now validated appliance, commanding it to perform two steps:

    a) Activate the services authorized for that subscriber.

    b) Calculate the RF fingerprint for the EUE at the present location and store it in persistent memory.

At periodic audit intervals, determined by EMM, the EUE shall collect data and calculate an electronic fingerprint value, comparing it to the reference value stored in memory. If the calculated value is within predetermined limits, no further action is taken until the next audit period.

Whenever the EUE is rebooted or otherwise reset, signifying a lapse in network connectivity where the device may have been relocated without the authorization of the cable operator, the EUE shall collect data and calculate an electronic fingerprint value, comparing it to the reference value stored in memory. If the calculated value is within predetermined limits, the device continues the booting process and services are transparently restored. If the match is unsuccessful, all television services shall be automatically self-deauthorized by the EUE and an on-screen message generated by the EUE and displayed on the subscriber's television screen indicating that the cable operator must be contacted at the AV&V telephone number contained within the message for EUE reactivation. This message occurs because the EUE has determined that an unauthorized relocation has occurred. When the subscriber calls the displayed telephone number, the AV&V process is started anew and the location of the device is re-evaluated.

Annex A — Part 1

Cable Emergency Alert System

The emergency alert system used in the digital converters shall comply with ANSI J-STD-042-2002 **[7]** . This system allows both one-way and two-way devices to receive emergency alerts and shall cause deterministic action to be taken at the EUE in response to valid alert messages.

Trigger Messages

EAS triggers may come in two different ways. The first method is through delivery of an emergency alert message carried within the MPEG transport multiplex on PID 0x1FFB. The message will be in the form of an MPEG section having a table_id value of 0xD8, as required by **[7]** . The second form of trigger delivery shall be through the EMM delivery system, in-band for one-way devices and through DOCSIS for two-way devices. These messages shall be carried within a unique EAS descriptor embedded in the specified EMM protocol. The complete structure of the emergency alert message section shall be conveyed within the EAS descriptor. Regardless of the method of delivery, all valid EAS messages shall be immediately parsed and the indicated action taken by the EUE. There shall be no preference or priority of one delivery method over the other and both paths shall be concurrently monitored by the EUE for message receipt.

Part 2 – Navigation Data Specification

Table of Contents

List of Figures – Part 2

List of Tables – Part 2

1 Scope

This specification provides a system definition for the navigation and other data tables used as part of the basic access control system. The intent of this document is to provide the detailed specifications necessary to implement both the headend/Source equipment and EUE device software enabled functionality.

The digital enduser terminal is an Enduser User Equipment device (EUE) utilizing publicly available open standards. For cable applications it shall receive digital broadcast services and descramble digitally encrypted services. For professional applications, the EUE may only receive MPEG-2 transport streams subjected to the Passage Process.

Two hardware versions are supported by this specification. The first supports only one-way real-time communication, such as a unidirectional digital cable television, COFDM or digital satellite receiver/decode. The control channel function is realized through in-band delivery to implement one-way connectivity, receiving control information and entitlement messages through open standard protocols. The other hardware version specified supports real-time two-way communications using DOCSIS or another return path to provide a two-way OOB control channel.

1.1 Overview

The purpose of the initiative, promoted by Sony, is to facilitate competition in an open marketplace by providing a means to deploy non-legacy source equipment, subscriber devices and services on existing legacy networks or distribution channels. The navigation data specification forms part of a tool kit allowing content owners and network operators to implement a method for digitally distributing content that was previously analog and to optionally choose alternative CA vendors while still preserving legacy choices. Migration paths to up-grades are also possible. Having the capability to support the coexistence of multiple CA systems without the need for content replication can be a bandwidth saver in many installations.

1.2 Reference Documents

Unless a specific version or revision number is included, the following references are non-specific, and the latest version applies.

[1] SP-RFIv1.1-I08-020301 DOCSIS v1.1, Radio Frequency Interface Specification, Cablelabs

[2] OC-SP-HOST-CFR-I10-020628, Open Cable Host Device Core Functional Requirements, Cablelabs

[3] ISO/IEC 13818-1:2000, Information Technology — Coding of Moving Pictures and Associated Audio — Part 1: Systems, International Telecommunications Union

[4] SCTE 54:2004, Digital Video Service Multiplex and Transport System Standard for Cable Television, Society of Cable Television Engineers

[5] ETSI EN 300 468 v1.4.1 (2000-11), Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB Systems

[6] EIA 542-A:2002, Cable Television Channel Identification Plan, Electronic Industry Association

[7] RFC 1112 [Deering, 1989], Host Extensions for IP Multicasting

[8] ANSI-J-STD-042-2002, Emergency Alert Message for Cable, ANSI/Society of Cable Television Engineers

1.3 Acronyms and Abbreviations

BAT	Bouquet Association Table
CA	Conditional Access
CASID	Conditional Access System Identifier
CMTS	Cable Modem Termination System
EUE	Customer Premises Equipment
DDC	Data Download Carousel
DHCP	Dynamic Host Configuration Protocol
DOCSIS	Data Over Cable Service Interface Specifications
DPI	Digital Program Insertion
DVB	Digital Video Broadcast
DVB-CSA	Digital Video Broadcast-Common Scrambling Algorithm
DVB-SI	Digital Video Broadcast-Service Information
DVR	Digital Video Recorder
EAS	Emergency Alert Service
EMM	Entitlement Management Message
EUE	End User Equipment (STB, IRD, Professional Decoder)
IGMP	Internet Group Management Protocol
IP	Internet Protocol (abbreviated form for TCP/IP)
MPEG	Moving Pictures Experts Group
MSO	Multiple System Operator
NIT	Network Information Table
NTSC	National Television Standards Committee
OOB	Out of Band
PAT	Program Allocation Table
PID	Packet Identifier
PMT	Program Map Table
PSI	Program Specific Information
QAM	Quadrature Amplitude Modulation
RAM	Random Access Memory
S-A	Scientific-Atlanta
SDT	Service Descriptor Table
SI	Service Information
STB	Set Top Box
TOT	Time Offset Table
TS	Transport Stream
UDP	User Datagram Protocol

2 EUE Network Provisioning

Regardless of whether a particular device is uni or bi-directional, all devices shall obtain basic network provisioning information in-band through a common provisioning message table carried in-band on all the digital overlay transport multiplexes. The provisioning message table carries information regarding the PIDs (for one-way devices) and IP addresses (for two-way devices) assigned to each SI table type and the download data carousel (DDC) message indicating the types and locations of EUE download images. The provisioning message is both low data rate and small size.

Each time the EUE boots, it shall access the provisioning message twice. The first access will occur from the SBS-Client contained in the bootloader in order to determine whether a EUE software upgrade is necessary. The second access occurs during initial execution of the EUE application to determine the applicable SI resources in the network. The actual values carried in the message may be cached along with the descriptor version in order to expedite the boot process and only parse the complete message if a table version change is detected.

The provisioning message shall be carried on each transport stream on a well-known PID. The default PID be defined as 0x1F00 and shall be invariant on the system. In any case, the EUE shall have a method for viewing/entering the provisioning message PID from the remote control/television display, allowing the default value set at the factory to be overwritten. Once entered, the value shall be stored in nonvolatile memory and used as the provisioning message PID until superseded by manual entry of another value.

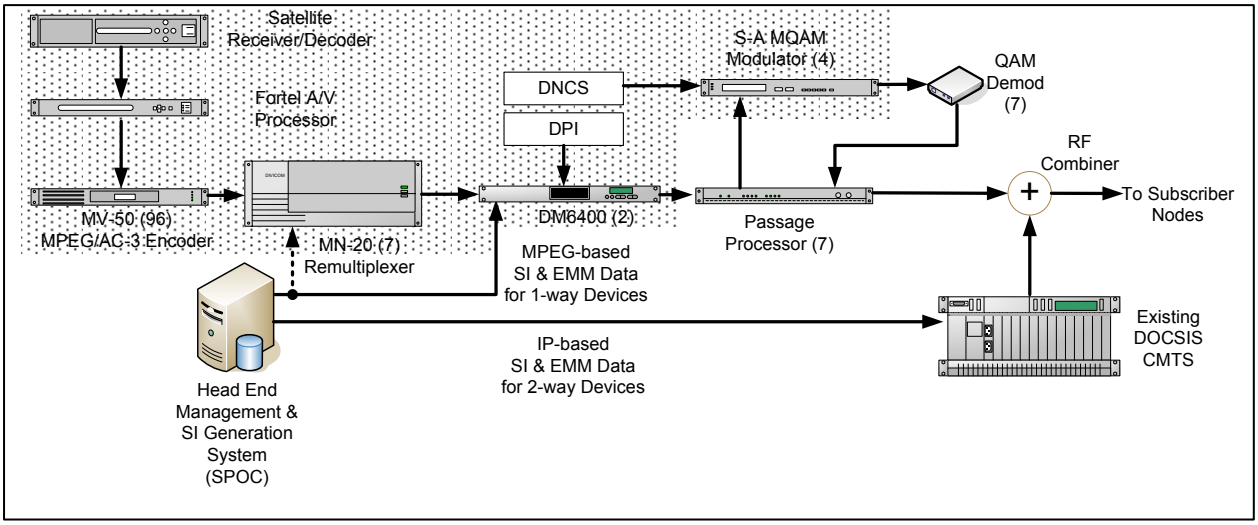


Figure 1 – System delivering both one-way signaling in-band and two-way signaling OOB via DOCSIS

2.1 One-way Networks

The configuration shown in Figure 1 shows a delivery system supporting both one-way and two-way EUE devices. In one-way configurations, the System Information (SI) and Entitlement Management Message (EMM) data are transferred from the headend/source to the set-top box via the in-band channel. There is no return communication from the set-top boxes to the headend/source.

2.1.1 In-band Navigation Data Delivery

In-band data such as NIT, BAT, TOT, SDT and CAT will be delivered using a data packetizer. This device is able to take an incoming IP stream and dynamically slice it into 184-byte data packets, adding the standard 4-byte

MPEG-2 header, which includes the assigned PID address. The outgoing packetized MPEG elementary stream is then sent to a remultiplexer for insertion into a transport stream. The same device can also locally store a data file and spool it indefinitely as a packetized MPEG-2 elementary stream. The packetizer is able to handle files up to 20Mbytes. Each SI section can be either specified as a separate file or delivered dynamically. Each file or stream can be specified to play out on the PID value assigned in ETSI-DVB-SI.

In-band data such as code images will be scrambled as part of its delivery. The code download is implemented via in-band MPEG-2 TS delivery as "private data" also using the same data packetizer. It uses the frequency and PID referenced in the transmitted MPEG-2 provisioning message to support this functionality.

CA messages such as EMM, if used, shall be delivered in-band using the packetizer. The messages shall follow the standard MPEG-2 protocol, being carried on the PID referenced under the **CA_PID** field of the CAT. The CAT follows the MPEG-2 format with the exception that the CAT shall not necessarily occur on the expected and standardized PID of 0x0001. Instead, it will be located on the PID identified within the private descriptor for the auxiliary CAT carried within the provisioning message and optionally within the NIT.

The system shall convey from the headend/source to the EUE client SI and entitlement information using standard MPEG-2 transport protocol and PID assignments, where applicable **[3]**. The PID assignments for SI tables may appear on any arbitrary PID except zero and 0x1FFF. Multiple tables and sections may be delivered on the same PID and it is up to the EUE device application to distinguish different tables and sections from the incoming data stream on each PID, as applicable.

The structure and format of each table shall remain identical to the corresponding table delivered via DOCSIS for two-way devices. The same EUE device parsing application shall be used to interpret the tables regardless of delivery method.

2.2 Two-way Networks

The configuration shown in Figure 1 shows a delivery system supporting both one-way and two-way EUE devices. In two-way configurations, the System Information (SI) and Entitlement Management Message (EMM) data are transferred from the headend/source to the set-top box via DOCSIS cable modem services. There is a return path available, for use in other applications.

2.2.1 DOCSIS-Based Navigation Data Delivery

The DOCSIS service will convey from the headend/source to the EUE client SI and entitlement information using the UDP multicast protocol. Multiple tables and sections may be delivered on the same multicast group and it is up to the EUE device application to distinguish different tables and sections from the incoming data stream on each multicast group, as applicable.

The structure and format of each table shall remain identical to the corresponding table delivered in-band for one-way devices. The same EUE device parsing application shall be used to interpret the tables regardless of delivery method. In the event that the two-way device cannot obtain a valid connection with the headend/source via DOCSIS after **1** minute, it shall automatically revert to in-band SI data and messaging reception until such time that DOCSIS connectivity is restored. In this one-way mode, navigation data provisioning will follow the same procedures as outlined in section 2.1.1 and operation, with respect to message reception and provisioning, will be indistinguishable from a one-way device during this period.

Upon obtaining a valid DOCSIS connection, signified by the A/V decoder application being assigned an IP address by the DHCP server, the EUE device application shall issue an IGMP join command to each of the multicast groups listed in the body of the IP provisioning descriptor, carried within the provisioning message, in order to begin reception of headend/source messaging.

IGMP messages are used by multicast routers to keep track of group membership on each of the router's physically attached networks. The following rules apply[1]:

1. A host sends an IGMP report when the first process joins a group. If multiple processes on a given host join the same group only one report is sent, the first time a process joins that group. This report is sent out the same interface on which the process joined the group.

2. A host does not send a report when processes leave a group, even when the last process leaves a group. The host knows that there are no members in a given group, so when it receives the next query (next step), it wont report the group.

3. A multicast router sends am IGMP query at regular intervals to see if any hosts still have processes belonging to any groups. The router must send one query out each interface. The group address in the query is 0 since the router expects one response from a host for every group that contains one or more members on that host.

4. A host responds to an IGMP query by sending one IGMP report fore each group that still contains at least one process.

3 Headend/Source Messages

3.1 Provisioning Message

Provisioning messages are mapped into MPEG-2 Transport packet payloads. The PUSI bit of the MPEG-2 Packet header being set indicates the start of a provisioning message. Subsequent packets on the same PID contain the remainder of the message. One message must complete before another may start. The provisioning message carries the basic information necessary for a EUE device, either one-way or two-way, to boot.

The provisioning message is carried as a private data stream and segmented into "MPEG-like" sections. Each provisioning message section shall not exceed 4096 bytes in length. If the total message payload exceeds 4096 bytes, then multiple sections shall be transmitted and the payload shall span sections, with the **section_number** and **last_section_number** indicating where a particular section fits in the sequence. In a multi-section provisioning message, each section does not terminate in a 4-byte CRC for each particular section.

Since descriptors defined in the provisioning message format each contain a signature or CRC, each descriptor (and message header) carries its own validation data internally and there is no composite CRC for the provisioning message. It is the responsibility of the receiving device to buffer a multi-section provisioning message in the correct sequential order and to remove any intervening section headers before provisioning message payload concatenation and descriptor CRC or signature validation. Because of the small data set sizes associated with the provisioning message, it is unlikely that a message would span many sections and the additional complexity of intermediate CRCs is therefore unwarranted. If an unexpected section header is encountered before reaching the data set size provided in **section_length** field during parsing of a received section in a multi-section message, the errant section should be discarded and re-received. If a descriptor fails to validate through evaluation of the signature or CRC, as appropriate, then the entire message should be re-received and the errant descriptor re-evaluated.

[1] TCP/IP Illustrated: The Protocols, W. Richard Stevens, 1994, Addison Wesley Longman

Table 1 – Provisioning Message Format

Syntax	No. of Bits	Mnemonic
provisioning_message_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
private_indicator	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
protocol_version	8	uimsbf
message_version	8	uimsbf
SI_provisioning_header(){		
reserved	3	bslbf
NIT_PID	13	uimsbf
reserved	3	bslbf
BAT_PID	13	uimsbf
reserved	3	bslbf
SDT_PID	13	uimsbf
reserved	3	bslbf
TOT_PID	13	uimsbf
reserved	3	bslbf
CAT_PID	13	uimsbf
header_version_number	8	uimsbf
}		
reserved	7	bslbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
CRC_32	32	rpchof
for (i=1; i<N;i++) {		
descriptor()		
}		
}		

table_id: An 8-bit field that shall be set to **0x86** to identify this private data table uniquely as a provisioning message.

section_syntax_indicator: A 1-bit field that shall be set to 0 to identify this table as a private section.

private_indicator: A 1-bit field that shall be set to 0 to identify the section payload as private data.

section_length: A 12-bit field that indicates the length of the entire section, with the count starting immediately following the ***section_length*** field, including all descriptors.

protocol_version: An 8-bit field indicating the version of the provision message section protocol.

message_version: An 8-bit field indicating the version of the provision message section. Each successively issued version of the section shall be unique and monotonically increasing.

header_version_number: An 8-bit field that shall that contains the version number of the ***SI_provisioning_header*** portion of the section. It shall be incremented by one whenever a change in the information carried within the header portion occurs.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in **[3]**, annex B after processing the entire span between the start of the table and the ***last_section_number***, inclusive.

The provisioning message carries one or more descriptors, which contain the actual provisioning payload. The different possible provisioning descriptors are:

- DDC Message — Described in Part 1 of this specification § 4.5.2
- Two-way IP Provisioning

3.1.1 Two-Way IP Provisioning Descriptor

The two-way IP provisioning descriptor is a private descriptor providing a two-way EUE device information regarding the network addresses of navigation information accessible through DOCSIS modem multicast services. Its format is defined in Table 2.

Table 2 – IP Provisioning Descriptor

Syntax	No. of Bits	Identifier
IP_provisioning_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
descriptor_version_number	5	uimsbf
reserved	3	bslbf
descriptor_body(){	32	
NIT_IP	16	uimsbf
NIT_port	32	uimsbf
BAT_IP	16	uimsbf
BAT_port	32	uimsbf
SDT_IP	16	uimsbf
SDT_port	32	uimsbf
TOT_IP	16	uimsbf
TOT_port	32	uimsbf
CAT_IP	16	uimsbf
CAT_port		uimsbf
}	32	
CRC-32		rpchof
}		

descriptor_tag: An 8-bit field that shall be set to **0xBE** to identify this descriptor uniquely as an IP provisioning descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the ***descriptor_length*** field, up to the end of this descriptor.

descriptor_version_number: A 5-bit field that shall that contains the version number of the **IP_provisioning_descriptor()**. It shall be incremented by one whenever a change in the information carried within the descriptor occurs.

NIT_IP,
BAT_IP,
SDT_IP,
TOT_IP,
CAT_IP: These 32-bit fields provide the tcp/ip or UDP addresses for each SI data service respectively. The data shall be organized as a packed set containing the four bytes as aa.bb.cc.dd.

NIT_port,
BAT_port,
SDT_port,
TOT_port,
CAT_port: These 16-bit fields provide the port addresses for each SI data service respectively.

CRC_32: This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in **[3]**, annex B after processing the entire descriptor.

3.2 Service Information (SI) Tables

This section does not define the SI table protocol. The Passage-enabled set-top box shall support all SI tables specified in the ETSI-DVB-SI specification.

Indicated for each table described in this document are the required and optional data fields and descriptors. Also described are any applicable private descriptors. While the descriptors likely to be encountered in a table are presented herein, this should not be construed as to preclude the appearance of other descriptors, public or private, in any table. The EUE device parsing the SI tables shall ignore any descriptors that it is not specifically tasked to interpret and in any case, the appearance of additional public or private descriptors shall not cause any abnormal operation of the device. Details regarding the structure, format and coding of the tables and all associated public data fields and descriptors are provided in ETSI EN 300 468 v1.4.1 (2000-11), *Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB Systems* **[5].**

3.2.1 Network Information Table (NIT) for Passage-enabled Networks

The NIT, as defined in ETSI-DVB **[5]** and restated in Table 3, carries information relating to the physical organization of the multiplexes/TSs carried via a given network, and of the characteristics of the network itself. The combination of **original_network_id** and **transport_stream_id** allow each transport to be uniquely identified.

Table 3 – Network Information Section

Syntax	No. of Bits	Identifier
Network_information_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
network_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_used	4	bslbf
network_descriptor_length	12	uimsbf
for (i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
transport_stream_loop_length	12	uimsbf
for (i=0;i<N;i++){		
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
reserved_future_use	4	bslbf
transport_descriptors_length	12	uimsbf
for (j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

Table 4 shows a tabular list of the required fields for the NIT data.

Table 4 – Network Information Section Required Fields

Location	Descriptor or Field	Public	Applicability
Header	**network_id**	Yes	Required
1st Loop	**network_name_descriptor**	Yes	Optional
	private_data_specifier_descriptor	Yes	Optional
End Loop			
2nd Loop	**transport_stream_id**	Yes	Required
	original_network_id	Yes	Required
	private_data_specifier_descriptor	Yes	Optional
	cable_delivery_system_descriptor	Yes	Required
	service_list_descriptor	Yes	Required
End Loop			

3.2.1.1 Cable Delivery System Descriptor

The cable delivery system descriptor is a public descriptor and provides a means to convey the location and associated parameters to access that particular transport in the cable system RF multiplex. The cable delivery system descriptor shall be listed in the second descriptor loop of the network information section. Its format is defined in **[5]** and Table 5 .

Table 5 – Cable Delivery System Descriptor

Syntax	No. of Bits	Identifier
cable_delivery_system_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
frequency	32	bslbf
reserved	12	bslbf
FEC_outer	4	bslbf
Modulation	8	bslbf
Symbol_rate	28	bslbf
FEC_inner	4	
}		

descriptor_tag: An 8-bit field that shall be set to 0x44 to uniquely identify this descriptor as a cable delivery system descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.

frequency: A 32-bit field giving the 4 bit BCD values specifying 8 characters of the frequency value, coded in MHz, where the decimal occurs after the fourth character (e.g., 0312.0000 MHz).

reserved: A padding field reserved for future definition and is retained only for format compliance with ETSI-DVB standards.

FEC_outer: This field carries 4 bits of undefined data and is retained only for format compliance with ETSI-DVB standards.

modulation: An 8-bit filed specifying the modulation scheme used on a cable delivery system according to reference **[5]**.

symbol_rate: This field carries 28 bits of undefined data and is retained only for format compliance with ETSI-DVB standards.

FEC_inner: This field carries 4 bits of undefined data and is retained only for format compliance with ETSI-DVB standards.

3.2.1.2 Service List Descriptor

The service list descriptor is a public descriptor providing a means of listing the services by **service_id** and **service_type**. The service list descriptor shall be listed in the second descriptor loop of the network information section. Its format is defined in **[5]** and Table 6.

Table 6 – Service List Descriptor

Syntax	No. of Bits	Identifier
service_list_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0;i<N;i++){		
service_id	16	uimsbf
service_type	8	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0x41 to uniquely identify this descriptor as a cable delivery system descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.

service_id: A 16-bit field which uniquely identifies a service within a TS.

service_type: An 8-bit field specifying the type of service, coded according to reference **[5]**.

3.2.2 Bouquet Association Table (BAT)

The BAT provides information regarding grouping of services and logical channel numbering. The BAT syntax in Table 7, as defined in ETSI-DVB **[5]**, is repeated here for the convenience of the implementer.

Table 7 – Bouquet Association Section

Syntax	No. of Bits	Identifier
bouquet_association_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
bouquet_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_used	4	bslbf
bouquet_descriptors_length	12	uimsbf
for (i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
transport_stream_loop_length	12	uimsbf
for (i=0;i<N;i++){		
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
reserved_future_use	4	bslbf
transport_descriptors_length	12	uimsbf
for (j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

Table 8 shows a tabular list of the required fields for BAT data.

Table 8 – Bouquet Association section Required Fields

Location	Descriptor or Field	Public	Applicability
Header	**bouquet_id**	Yes	Required
1st Loop	**private_data_specifier_descriptor**	Yes	Optional
	bouquet_name_descriptor	Yes	Optional
	multilingual_bouquet_name_descriptor	Yes	Optional
End Loop			
2nd Loop	**transport_stream_id**	Yes	Required
	original_network_id	Yes	Required
	service_list_descriptor	Yes	Required
	private_data_specifier_descriptor	Yes	Optional
	logical_channel_descriptor	No	Optional
End Loop			

3.2.2.1 Service List Descriptor

The service list descriptor is a public descriptor providing a means of listing the services by **service_id** and **service_type**. The service list descriptor shall be listed in the second descriptor loop of the bouquet association section. Its format is defined in **[5]** and Table 6.

3.2.2.2 Logical Channel Descriptor

The logical channel descriptor is a private descriptor that provides a means of correlating services by **service_id** to a **logical_channel_number** so that channel numbers, arbitrarily assigned by the cable system operator, can be uniquely mapped to a specific service, irrespective of actual **service_id**. The service list descriptor shall be listed in the second descriptor loop of the network information section and/or bouquet association section. Its format is defined in table 9.

Table 9 – Logical Channel Descriptor

Syntax	No. of Bits	Identifier
logical_channel_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0;i<N;i++){		
service_id	16	uimsbf
logical_channel_number	16	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xE2 to uniquely identify this descriptor as a cable delivery system descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.

service_id: A 16-bit field which uniquely identifies a service within a TS.

Logical_channel_number: A 16-bit field specifying the logical number associated with the immediately preceding **service_id**.

3.2.3 Service Description Table (SDT)

The SDT describes the services in the system, such as service names and providers. Its format is defined in ETSI-DVB **[5]** and repeated in Table 10.

Table 10 – Service Description Section

Syntax	No. of Bits	Identifier
service_description_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
original_network_id	16	uimsbf
reserved_future_used	8	bslbf
for (i=0;i<N;i++){		
service_id	16	uimsbf
reserved_future_use	6	bslbf
EIT_schedule_flag	1	bslbf
EIT_present_following_flag	1	bslbf
running_status	3	uimsbf
free_CA_mode	1	bslbf
descriptors_loop_length	12	uimsbf
for (j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

Table 11 shows a tabular list of the required fields for SDT data.

Table 11 – Service Description Section Required Fields

Location	Descriptor or Field	Public	Applicability
Header	**transport_stream_id**	Yes	Required
	original_network_id	Yes	Required
1st Loop	**service_id**	Yes	Required
	EIT_schedule_flag	Yes	Padding Bits
	EIT_schedule_present_following_flag	Yes	Padding Bits
	running_status	Yes	Padding Bits
	free_CA_mode	Yes	Padding Bits
	private_data_specifier_descriptor	Yes	Optional
	service_descriptor	Yes	Required
	multilingual_service_name_descriptor	Yes	Optional
	Extended_service_descriptor	No	Required
End Loop			

3.2.3.1 Service Descriptor

The service descriptor is a public descriptor providing the names of the service provider and service in text form together with the **service_type**. Its format is defined in **[5]** and Table 12.

Table 12 – Service Descriptor

Syntax	No. of Bits	Identifier
service_list_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
service_type	8	uimsbf
service_provider_name_length	8	uimsbf
for (i=0;i<N;i++){		
char	8	uimsbf
}		
service_name_length	8	uimsbf
for (i=0;i<N;i++){		
char	8	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0x48 to uniquely identify this descriptor as a service descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.

service_type: An 8-bit field specifying the type of service, coded according to reference **[5]**.

service_provider_name_length: An 8-bit field specifying the string length, in bytes, of the service provider name string.

char: An 8-bit field making up a string of text coded according to reference **[5]**.

service_name_length: An 8-bit field specifying the string length, in bytes, of the service name character string.

char: An 8-bit field making up a string of text coded according to reference **[5]**.

3.2.3.2 Extended Service Descriptor

The service descriptor is a private descriptor providing the names of the service provider and service in text form together with the **service_type**. Its format is defined in Table 13.

Table 13 – Extended Service Descriptor

Syntax	No. of Bits	Identifier
extended_service_list_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
service_type	8	uimsbf
abbreviated_service_provider_name_length	8	uimsbf
for (i=0;i<N;i++){		
char	8	uimsbf
}		
Abbreviated_service_name_length	8	uimsbf
for (i=0;i<N;i++){		
char	8	uimsbf
}		
}		

descriptor_tag: An 8-bit field that shall be set to 0xC0 to uniquely identify this descriptor as an extended service descriptor.

descriptor_length: An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.

service_type: An 8-bit field specifying the type of service, coded according to reference **[5]**.

abbreviated_service_provider_name_length: An 8-bit field specifying the string length, in bytes, of an abbreviated form of the service provider name string.

char: An 8-bit field making up a string of text coded according to reference **[5]**.

abbreviated_service_name_length: An 8-bit field specifying the string length, in bytes, of an abbreviated form of the service name character string.

char: An 8-bit field making up a string of text coded according to reference **[5]**.

3.2.4 Time Offset Table (TOT)

The Time Offset Table (TOT) carries information relating to the present time and date and local time offset from UTC. The TOT format is defined in ETSI-DVB **[5]** and Table 14.

Table 14 – Time Offset Section

Syntax	No. of Bits	Identifier
time_offset_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
UTC_time	40	bslbf
reserved	4	bslbf
descriptors_loop_length	12	uimsbf
for (i=0;i<N;i++){		
descriptor()		
}		
CRC_32	32	rpchof
}		

Table 15 shows a tabular list of the required fields for TOT data.

Table 15 – Time Offset Section Required Fields

Location	Descriptor or Field	Public	Applicability
Header	**UTC_time**	Yes	Required
1st Loop	**Local_time_offset_descriptor**	Yes	Required
End Loop			

3.2.4.1 Local Time Offset Descriptor

The service descriptor is a public descriptor providing the names of the service provider and service in text form together with the service_type. Its format is defined in **[5]** and Table 16.

Table 16 – Local Time Offset Descriptor

Syntax	No. of Bits	Identifier
local_time_offset_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for (i=0;i<N;i++){		
country_code	24	bslbf
country_region_id	6	bslbf
reserved	1	bslbf
local_time_offset_polarity	1	bslbf
local_time_offset	16	bslbf
time_of_change	40	bslbf
nest_time_offset	16	bslbf
}		
}		

descriptor_tag:	An 8-bit field that shall be set to 0x58 to uniquely identify this descriptor as a service descriptor.
descriptor_length:	An 8-bit field specifying the number of bytes immediately following the **descriptor_length** field, up to the end of this descriptor.
country_code:	A 24-bit field specifying the 3-character country code, coded according to reference **[5]**.
country_region_id:	A 6-bit field specifying the applicable time zone of the service area.
reserved:	A padding field reserved for future definition and is retained only for format compliance with ETSI-DVB standards.
local_time _offset_polarity:	A 1-bit field specifying the polarity of the time offset from UTC.
local_time_offset:	A 16-bit field containing the current offset in hours from UTC for the applicable service area, coded according to reference **[5]**.
time_of_change:	A 40-bit field specifying the date and time of the next time offset change, such as daylight savings time beginning or end and is coded according to reference **[5]**.
next_time_offset:	A 16-bit field containing the next applicable offset in hours from UTC for the applicable service area, coded according to reference **[5]**.

3.3 Program Specific Information

This section defines the Program Specific Information (PSI) delivered in the Passage enabled EUE. The PSI follows the ISO/IEC standard **[3]** , which contains both the public and private data that a Passage enabled EUE devices needs in order to access Passage processed services within the transport. The PSI and all its tables are delivered in the clear.

3.3.1 Program Association Table

The PAT follows the ISO/IEC PAT syntax. It provides the correspondence between a program number and the Passage program PID value of the transport stream packets, which carry the program definition.

Table 17 is a restatement of the Program Association Table defined in **[3]**. The information is repeated here for the convenience of the implementer.

Table 17 – Program Association Section

Syntax	No. of Bits	Mnemonic
program_association_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	Uimsbf
for (i=0; i<N;i++) {		
program_number	16	Uimsbf
reserved	3	Bslbf
if (program_number=='0') {		
network_PID	13	Uimsbf
}		
else {		
program_map_PID	13	Uimsbf
}		
}		
CRC_32	32	rpchof
}		

3.3.2 Program Map Table

The PMT follows the ISO/IEC PMT syntax. It provides the mapping between a program numbers and the PID values for both the elementary streams.

Table 18 is a restatement of the Program Map Table defined in **[3]**. The information is repeated here for the convenience of the implementer.

Table 18 – Program Map Table

Syntax	No. of Bits	Mnemonic
TS_program_map_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved	3	bslbf
PCR_PID	13	uimsbf
reserved	4	bslbf
program_info_length	12	Uimsbf
for (i=0; i<N;i++) {		
descriptor()		
}		
for (i=0; i<N1;i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID	13	uimsbf
reserved	4	bslbf
ES_info_length	12	Uimsbf
for (i=0; i<N2;i++) {		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

3.3.3 CA descriptors within the PMT

CA system ID value shall be 0x0001 for BAC implementations without an alternative CAS.

3.3.4 Conditional Access Table

The CAT follows the same syntax as ISO/IEC. It provides the mapping between a particular conditional access system and the PID value of the stream containing messages unique to that specific CA system.

Table 19 is a restatement of the Conditional Access Table defined in **[3]**. The information is repeated here for the convenience of the implementer.

Table 19 – Conditional Access Table

Syntax	No. of Bits	Mnemonic
CA_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
reserved	18	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
for (i=0; i<N;i++) {		
descriptor()		
}		
CRC_32	32	rpchof
}		

3.3.5 CA descriptors within CAT

The CA descriptor in the CAT follows the format given in Table 20 . In order to provide for the same data structure to be carried in both the in-band and DOCSIS SI distribution systems, the descriptor has defined two additional parameters that occur in the "**private_data_byte**" area provided in the MPEG-2 specification **[3]**. These two parameters convey the IP address and port number, respectively, for the EMM messages on devices accessing SI delivery using DOCSIS carriage. The CA PID referenced in the MPEG-2 defined descriptor shall be used to indicate the PID used for EMM transmission on the transport in which the particular CAT is received. The scope of any particular CAT table is constrained to the transport in which it occurs because the MPEG-2 specification provides for the possibility of each transport to have differing CA PIDS. In the DOCSIS transmission case, there is only a single broadcast channel (IP) carrying EMM data. Therefore, while the PIDs referenced in the CAT on various transports may be different, the EMM IP address will be constant in CAT sections received on all transports for a given system. For the CAT table received through DOCSIS transmission, the receiving EUE will ignore the PID information and use only the EMM provisioning information contained at the end of the descriptor.

Table 20 – Conditional Access Descriptor

Syntax	No. of Bits	Identifier
CA_descriptor(){		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
CA_system_ID	16	uimsbf
reserved	3	bslbf
CA_PID	13	uimsbf
EMM_IP	32	uimsbf
EMM_port	16	uimsbf
}		
}		

3.4 Miscellaneous Headend/Source Messages

This section covers messaging that is not otherwise standardized in the published documents. Per section 2.2 and 2.3, regardless of delivery method the content and format shall be identical.

3.4.1 Entitlement Management Messages

Entitlement management messages (EMMs), applicable in all phases, shall be received and interpreted by the EUE device per the specification provided in part one of this document.

3.4.2 Emergency Alert Messages

Emergency alert messages, where such delivery is mandated or desired, are delivered through a variety of paths and are all concurrently active. The data format and meaning is defined in **[8]**.

3.4.3 Code Download Messages

A software download message, carried within the EMM, shall be received and interpreted by the EUE device. The data payload format and meaning is given in part one of this document.