

ETSI TS 103 197 V1.5.1 (2008-10)

Technical Specification

Digital Video Broadcasting (DVB); Head-end implementation of DVB SimulCrypt

European Broadcasting Union



Union Européenne de Radio-Télévision



Reference

RTS/JTC-DVB-231

Keywords

broadcasting, digital, DVB, interface, video

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.

Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, please send your comment to one of the following services:

http://portal.etsi.org/chaicor/ETSI_support.asp

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2008.

© European Broadcasting Union 2008.

All rights reserved.

DECTTM, **PLUGTESTS**TM, **UMTS**TM, **TIPHON**TM, the TIPHON logo and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.

3GPPTM is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

Contents

Intellectual Property Rights	12
Foreword.....	12
1 Scope	13
1.1 Common scrambling algorithm	13
1.2 Language	13
2 References	13
2.1 Normative references	14
2.2 Informative references.....	15
3 Definitions and abbreviations.....	16
3.1 Definitions	16
3.2 Abbreviations	18
4 Architecture.....	19
4.1 System architecture	19
4.1.1 Host Head-end components	20
4.1.2 Simulcrypt CA components.....	20
4.1.3 Simulcrypt Integrated Management Framework (SIMF).....	21
4.1.4 Multiplexer Redundancy.....	21
4.2 Description of Components.....	21
4.2.1 Event Information Scheduler (EIS)	21
4.2.2 Simulcrypt Synchronizer (SCS).....	21
4.2.3 ECM Generator (ECMG).....	21
4.2.4 EMM Generator (EMMG).....	22
4.2.5 Private Data Generator (PDG).....	22
4.2.6 Service Information Generator (SIG)	22
4.2.7 Program Specific Information Generator (PSIG)	22
4.2.8 Custom Service Information Generator (CSIG)	22
4.2.9 Custom Program Specific Information Generator (CPSIG)	22
4.2.10 Multiplexer Configuration (MUXCONFIG)	22
4.2.11 Multiplexer (MUX).....	23
4.2.12 Scrambler (SCR).....	23
4.2.13 Control Word Generator (CWG)	23
4.2.14 Network Management System (NMS).....	23
4.2.15 SIMF agent	23
4.2.16 Access Criteria Generator (ACG).....	23
4.3 Description of interfaces	23
4.3.1 ECMG \Leftrightarrow SCS	23
4.3.2 EMMG \Leftrightarrow MUX	23
4.3.3 PDG \Leftrightarrow MUX.....	23
4.3.4 Custom (P)SI Generator \Leftrightarrow (P)SI Generator	24
4.3.5 EIS \Leftrightarrow SIG.....	24
4.3.6 (P)SI Generator \Leftrightarrow MUX.....	24
4.3.7 EIS \Leftrightarrow MUXCONFIG	24
4.3.8 MUXCONFIG \Leftrightarrow PSIG	24
4.3.9 MUXCONFIG \Leftrightarrow SCS	24
4.3.10 MUX \Leftrightarrow SCR	24
4.3.11 SCR onward.....	24
4.3.12 SCS \Leftrightarrow MUX.....	24
4.3.13 SCS \Leftrightarrow SCR	24
4.3.14 SCS \Leftrightarrow CWG.....	24
4.3.15 EIS \Leftrightarrow SCS	24
4.3.16 ACG \Leftrightarrow EIS.....	25
4.3.17 NMS Component \Leftrightarrow SIMF Agent.....	25
4.3.18 SIMCOMP \Leftrightarrow MUXCONFIG.....	25

4.3.19	Mandatory or optional characteristics of Simulcrypt interfaces	25
4.4	Protocol types	26
4.4.1	Connection-oriented TLV protocols	26
4.4.2	Connection oriented XML protocols	29
4.4.3	SIMF-based protocols.....	29
5	ECMG \Leftrightarrow SCS interface	29
5.1	Interface principles	29
5.1.1	Channel and Stream specific messages.....	29
5.1.2	Channel establishment	30
5.1.3	Stream establishment	30
5.1.4	Stream closure	30
5.1.5	Channel closure	30
5.1.6	Channel/Stream testing and status	30
5.1.7	Unexpected communication loss	31
5.1.8	Handling data inconsistencies.....	31
5.2	Parameter_type values.....	31
5.3	Parameter semantics	32
5.4	Channel specific Messages.....	34
5.4.1	Channel_setup message: ECMG \Leftarrow SCS.....	34
5.4.2	Channel_test message: ECMG \Leftrightarrow SCS	34
5.4.3	Channel_status message: ECMG \Leftrightarrow SCS.....	34
5.4.4	Channel_close message: ECMG \Leftarrow SCS	35
5.4.5	Channel_error message: ECMG \Leftrightarrow SCS	35
5.5	Stream specific messages	35
5.5.1	Stream_setup message: ECMG \Leftarrow SCS.....	35
5.5.2	Stream_test message: ECMG \Leftrightarrow SCS	35
5.5.3	Stream_status message: ECMG \Leftrightarrow SCS.....	35
5.5.4	Stream_close_request message: ECMG \Leftarrow SCS	36
5.5.5	Stream_close_response message: ECMG \Rightarrow SCS	36
5.5.6	Stream_error message: ECMG \Leftrightarrow SCS	36
5.5.7	CW_provision message: ECMG \Leftarrow SCS.....	36
5.5.8	ECM_response message: ECMG \Rightarrow SCS.....	38
5.6	Error status	38
5.7	Security in ECMG \Leftrightarrow SCS protocol.....	39
6	EMMG \Leftrightarrow MUX and PDG \Leftrightarrow MUX interfaces	39
6.1	Transport layer protocols for EMMG/PDG \Leftrightarrow MUX interfaces	39
6.2	TCP-based protocol.....	40
6.2.1	Interface principles	40
6.2.1.1	Channel and Stream specific messages.....	40
6.2.1.2	Channel establishment	40
6.2.1.3	Stream establishment	40
6.2.1.4	Bandwidth allocation	41
6.2.1.5	Stream closure.....	41
6.2.1.6	Channel closure.....	41
6.2.1.7	Channel/Stream testing and status.....	41
6.2.1.8	Unexpected connection loss	41
6.2.1.9	Handling data inconsistencies	41
6.2.2	Parameter Type Values	42
6.2.3	Parameter semantics	42
6.2.4	Channel specific messages.....	43
6.2.4.1	Channel_setup message: EMMG/PDG \Rightarrow MUX.....	43
6.2.4.2	Channel_test message: EMMG/PDG \Leftrightarrow MUX	43
6.2.4.3	Channel_status message: EMMG/PDG \Leftrightarrow MUX.....	43
6.2.4.4	Channel_close message: EMMG/PDG \Rightarrow MUX	44
6.2.4.5	Channel_error message: EMMG/PDG \Leftrightarrow MUX	44
6.2.5	Stream specific messages.....	44
6.2.5.1	Stream_setup message: EMMG/PDG \Rightarrow MUX.....	44
6.2.5.2	Stream_test message: EMMG/PDG \Leftrightarrow MUX	44
6.2.5.3	Stream_status message: EMMG/PDG \Leftrightarrow MUX.....	44

6.2.5.4	Stream_close_request message: EMMG/PDG \Rightarrow MUX.....	45
6.2.5.5	Stream_close_response message: EMMG/PDG \Leftarrow MUX	45
6.2.5.6	Stream_error message: EMMG/PDG \Leftrightarrow MUX	45
6.2.5.7	Stream_BW_request message: EMMG/PDG \Rightarrow MUX	45
6.2.5.8	Stream_BW_allocation message: EMMG/PDG \Leftarrow MUX	46
6.2.5.9	Data_provision message: EMMG/PDG \Rightarrow MUX.....	46
6.2.6	Error status.....	46
6.3	UDP-based protocol.....	47
6.3.1	Interface principles	47
6.3.1.1	Data_provision message: EMMG/PDG \Rightarrow MUX.....	48
6.3.1.2	Channel and stream configuration messages.....	49
6.3.2	Bandwidth management	49
7	Network management.....	49
7.1	SIMF overview.....	49
7.1.1	Introduction to the Common Information Model (CIM)	50
7.1.2	SIMF specialization options	51
7.2	Common Information Model (CIM).....	51
7.2.1	Object Containment Hierarchy	52
7.2.2	MIB II.....	54
7.2.3	Concurrency control	54
7.2.4	Simulcrypt Events Module (SEM).....	55
7.2.4.1	Event Group	57
7.2.4.2	Event Forwarding Discriminator (EFD) Group	58
7.2.4.3	Event Notification Group.....	59
7.2.4.4	Conformance requirements	60
7.2.5	Simulcrypt Logs Module (SLM)	62
7.2.5.1	Log Control Group.....	64
7.2.5.2	Logs Group	66
7.2.5.3	Conformance Requirements.....	67
7.3	CAS component monitoring and configuration.....	69
7.3.1	Ident Group.....	71
7.3.2	ECM Generator Group.....	71
7.3.3	EMMG/PDG Group.....	73
7.3.4	C(P)SIG Group	76
7.3.5	Conformance Requirements.....	79
8	C(P)SIG \Leftrightarrow (P)SIG interface.....	79
8.1	Overview and Scope.....	79
8.1.1	Note on commercial agreements.....	80
8.1.2	Note on the PDG \Leftrightarrow MUX Interface	80
8.2	Application protocol model.....	81
8.2.1	Overview of the C(P)SIG \Leftrightarrow (P)SIG Application Protocol.....	81
8.2.2	Configurations and Topologies.....	81
8.2.3	Trigger Transaction Type	82
8.2.4	Table Provisioning Transaction Type	84
8.2.5	Descriptor Insertion Transaction Type	85
8.2.6	Service Change Transaction Type	87
8.2.7	Flow PID Provisioning Transaction Type	88
8.2.8	Implementation of the C(P)SIG \Leftrightarrow (P)SIG protocol	91
8.3	Connection-oriented protocol.....	91
8.3.1	Overview of the C(P)SIG \Leftrightarrow (P)SIG connection-oriented protocol	91
8.3.1.1	Principles.....	91
8.3.1.2	Channels.....	92
8.3.1.2.1	Definition and types	92
8.3.1.2.2	Channel establishment.....	92
8.3.1.3	Streams.....	92
8.3.1.3.1	Definition.....	92
8.3.1.3.2	Stream establishment.....	93
8.3.1.4	C(P)SIG \Leftrightarrow (P)SIG message lists	93
8.3.1.5	Protocol state machines definition	94
8.3.1.6	Channel state machine.....	94

8.3.1.6.1	Channel Not Open	95
8.3.1.6.2	Channel Setting Up	95
8.3.1.6.3	Channel Open	95
8.3.1.6.4	Channel In Error	96
8.3.1.7	Stream state machine	96
8.3.1.7.1	Stream Not Open	98
8.3.1.7.2	Stream Setting Up	98
8.3.1.7.3	Stream Open	98
8.3.1.7.4	Stream Trigger Enabling	99
8.3.1.7.5	Stream Trigger-Enabled	99
8.3.1.7.6	Stream In Error	100
8.3.1.7.7	Stream Closing	100
8.3.1.8	Summary of messages permissible in each state	100
8.3.2	C(P)SIG \Leftrightarrow (P)SIG message syntax and semantics	102
8.3.2.1	List of message parameters for the C(P)SIG \Leftrightarrow (P)SIG protocol	102
8.3.2.2	Parameter semantics	103
8.3.3	Channel-level messages	107
8.3.3.1	Channel_setup message: C(P)SIG \Leftarrow (P)SIG	107
8.3.3.2	Channel_status message: C(P)SIG \Leftrightarrow (P)SIG	107
8.3.3.3	Channel_test message: C(P)SIG \Leftrightarrow (P)SIG	108
8.3.3.4	Channel_close message: C(P)SIG \Leftarrow (P)SIG	108
8.3.3.5	Channel_error message: C(P)SIG \Leftrightarrow (P)SIG	108
8.3.4	Stream-level messages	109
8.3.4.1	stream_setup message: C(P)SIG \Leftarrow (P)SIG	109
8.3.4.2	Stream_status message: C(P)SIG \Leftrightarrow (P)SIG	109
8.3.4.3	Stream_test message: C(P)SIG \Leftrightarrow (P)SIG	110
8.3.4.4	Stream_close message: C(P)SIG \Leftarrow (P)SIG	110
8.3.4.5	Stream_close_request message: C(P)SIG \Rightarrow (P)SIG	110
8.3.4.6	Stream_close_response message: C(P)SIG \Leftarrow (P)SIG	110
8.3.4.7	Stream_error message: C(P)SIG \Leftrightarrow (P)SIG	111
8.3.4.8	Stream_service_change message: C(P)SIG \Leftarrow (P)SIG	111
8.3.4.9	Stream_trigger_enable_request message: C(P)SIG \Rightarrow (P)SIG	111
8.3.4.10	Stream_trigger_enable_response message: C(P)SIG \Leftarrow (P)SIG	112
8.3.4.11	Trigger message: C(P)SIG \Leftarrow (P)SIG	112
8.3.4.12	Table_request message: C(P)SIG \Rightarrow (P)SIG	113
8.3.4.13	Table_response message: C(P)SIG \Leftarrow (P)SIG	114
8.3.4.14	Descriptor_insert_request message: C(P)SIG \Rightarrow (P)SIG	114
8.3.4.15	Descriptor_insert_response message: C(P)SIG \Leftarrow (P)SIG	115
8.3.4.16	PID_provision_request message: C(P)SIG \Rightarrow (P)SIG	116
8.3.4.17	PID_provision_response message: C(P)SIG \Leftarrow (P)SIG	116
8.3.5	Error status and error information	117
8.4	SIMF-based protocol	118
8.4.1	Operations Reference Points (ORPs)	118
8.4.2	Application of ORPs to the C(P)SIG \Leftrightarrow (P)SIG Interface	119
8.4.2.1	ECM/Event/Flow Change Triggering	120
8.4.2.2	(P)SI Table Provisioning	120
8.4.2.3	(P)SI Descriptor Insertion	120
8.4.2.4	Transport Stream Service Changes	121
8.4.2.5	PID Provisioning	121
8.4.3	SIM (P)SIG Group Specification	121
8.4.3.1	Information Table	121
8.4.3.2	Configuration Table	122
8.4.3.3	ECM Trigger Table	122
8.4.3.4	Flow PID Change Trigger Table	123
8.4.3.5	Event Trigger Table	124
8.4.3.6	PD Trigger Table	125
8.4.3.7	Descriptor Insert Table	126
8.4.3.8	Descriptor Insert Descriptor Table	128
8.4.3.9	Table Request Table	128
8.4.3.10	PID Provisioning Table	130

8.4.4	Conformance Requirements.....	130
9	(P)SIG \Leftrightarrow MUX interface	131
9.1	Overview	131
9.2	Interface principles	132
9.2.1	Description.....	132
9.2.1.1	Model of the interface (P)SIG \Leftrightarrow MUX with the carousel built in the MUX.....	132
9.2.1.2	Model of the interface (P)SIG \Leftrightarrow MUX with the carousel built in the (P)SIG.....	133
9.2.2	Channel and Stream specific messages.....	133
9.2.3	Channel establishment	134
9.2.4	Stream level protocol for the model with the carousel in the MUX	134
9.2.4.1	Stream establishment	134
9.2.4.2	Provision of the PSI/SI or private sections.....	135
9.2.4.3	Stream closure.....	135
9.2.5	Stream level protocol for the model with the carousel in the (P)SIG	135
9.2.5.1	Stream establishment	135
9.2.5.2	Bandwidth allocation	136
9.2.5.3	Stream closure.....	136
9.2.6	Channel closure	136
9.2.7	Channel/Stream testing and status	136
9.2.8	Unexpected communication loss	136
9.2.9	Handling data inconsistencies.....	136
9.2.10	Error management.....	137
9.3	Parameter_type values.....	137
9.4	Parameter semantics	137
9.5	Channel specific Messages.....	139
9.5.1	Channel_setup message: (P)SIG \Rightarrow MUX.....	139
9.5.2	channel_test message: (P)SIG \Leftrightarrow MUX	139
9.5.3	channel_status message: (P)SIG \Leftrightarrow MUX.....	139
9.5.4	channel_close message: (P)SIG \Rightarrow MUX.....	140
9.5.5	channel_error message: (P)SIG \Leftrightarrow MUX.....	140
9.6	Stream specific messages for both models	140
9.6.1	stream_setup message: (P)SIG \Rightarrow MUX	140
9.6.2	Stream_test message: (P)SIG \Leftrightarrow MUX	140
9.6.3	Stream_status message: (P)SIG \Leftrightarrow MUX.....	141
9.6.4	Stream_close_request message: (P)SIG \Rightarrow MUX	141
9.6.5	Stream_close_response message: (P)SIG \Leftarrow MUX.....	141
9.6.6	Stream_error message: (P)SIG \Leftrightarrow MUX.....	141
9.7	Specific messages for the model with the carousel in the MUX	142
9.7.1	CiM_stream_section_provision: (P)SIG \Rightarrow MUX.....	142
9.7.2	CiM_channel_reset: (P)SIG \Rightarrow MUX.....	142
9.8	Specific messages for the model with the carousel in the (P)SIG	143
9.8.1	CiP_Stream_BW_request message: (P)SIG \Rightarrow MUX	143
9.8.2	CiP_stream_BW_allocation message: (P)SIG \Leftarrow MUX.....	143
9.8.3	CiP_stream_data_provision message: (P)SIG \Rightarrow MUX	143
9.9	Error status	143
10	EIS \Leftrightarrow SCS Interface	144
10.1	Overview	144
10.2	Interface principles	145
10.2.1	Channel specific messages.....	145
10.2.2	Scrambling Control Group (SCG) specific messages	145
10.2.3	Channel establishment	146
10.2.4	Scrambling Control Group provisioning.....	146
10.2.5	Channel closure	146
10.2.6	Channel testing and status.....	146
10.2.7	Scrambling Control Group testing and status	147
10.2.8	Unexpected communication loss	147
10.2.9	Handling data inconsistencies.....	147
10.2.10	Error management.....	147
10.3	Parameter_type values.....	148

10.4	Parameter Semantics	148
10.5	Channel specific messages	150
10.5.1	channel_setup message: EIS \Rightarrow SCS	150
10.5.2	channel_test message: EIS \Leftrightarrow SCS	150
10.5.3	channel_status message: EIS \Leftrightarrow SCS	150
10.5.4	channel_close message: EIS \Rightarrow SCS	151
10.5.5	channel_reset message: EIS \Rightarrow SCS	151
10.5.6	channel_error message: EIS \Leftrightarrow SCS	151
10.6	SCG specific messages	152
10.6.1	SCG_provision message: EIS \Rightarrow SCS	152
10.6.2	SCG_test message: EIS \Rightarrow SCS	153
10.6.3	SCG_status message: EIS \Leftarrow SCS	154
10.6.3.1	Response to a provisioning message	154
10.6.3.2	Response to a test message	154
10.6.3.3	Management of the <i>SCG_nominal_CP_duration</i> parameter	154
10.6.4	SCG_list_request message: EIS \Rightarrow SCS	155
10.6.5	SCG_list_response message: EIS \Leftarrow SCS	155
10.6.6	SCG_error message: EIS \Leftarrow SCS	155
10.6.7	ECM_Group: CompoundTLV	155
10.7	Error status	156
11	ACG \Leftrightarrow EIS Interface	157
11.1	Overview	157
11.2	Scope	157
11.2.1	Role of the ACG	157
11.2.2	Role of the EIS	158
11.2.3	Dynamics of the ACG \Leftrightarrow EIS Interface	159
11.3	Interface Principles	160
11.3.1	General Principles	160
11.3.2	Channel Specific Messages	160
11.3.3	Messages for Access Criteria Creation and Modification	161
11.3.4	Channel Establishment	161
11.3.5	Channel Closure	161
11.3.6	Channel Testing and Status	161
11.3.7	Unexpected Communication Loss	161
11.3.8	Handling Data Inconsistencies	161
11.3.9	Handling Multiple AC Parameters in one AC Response	161
11.3.10	Handling AC Expiration Time	162
11.3.11	Asynchronous Access Criteria Change Request	162
11.3.12	Inserting Additional Information in the AC	163
11.3.13	Data Communications and Message Format	163
11.4	Interface Structure	163
11.5	Parameter Semantics	165
11.6	Parameter Types	167
11.7	Parameters Substitution	167
11.8	Channel Specific Messages	168
11.8.1	The <i>channel_setup</i> Message (EIS \Rightarrow ACG)	168
11.8.2	The <i>channel_test</i> Message (EIS \Leftrightarrow ACG)	169
11.8.3	The <i>channel_status</i> Message (EIS \Leftrightarrow ACG)	169
11.8.4	The <i>channel_error</i> Message (EIS \Leftrightarrow ACG)	170
11.8.5	The <i>channel_close</i> Message (EIS \Rightarrow ACG)	170
11.9	Messages of Access Criteria Creation and Modification	170
11.9.1	The <i>AC_request</i> Message (EIS \Rightarrow ACG)	170
11.9.2	The <i>AC_response</i> Message (EIS \Leftarrow ACG)	171
11.9.3	The <i>AC_interrupt</i> Message (EIS \Leftarrow ACG)	172
11.9.4	The <i>AC_error</i> Message (EIS \Leftarrow ACG)	172
11.10	Error Status	173
12	SIMCOMP \Leftrightarrow MUXCONFIG Interface	173
12.1	Overview	173
12.2	Interface Principles	174

12.2.1	System Diagram.....	174
12.2.2	Data Communications and Message Format	175
12.2.3	Message Groups.....	175
12.2.3.1	Communication channel messages.....	175
12.2.3.2	Transport resource mapping messages.....	175
12.2.4	Channel Establishment	176
12.2.5	Timeout and Retry	176
12.2.6	Channel closure	176
12.2.7	Channel testing and status.....	177
12.2.8	Handling data inconsistencies.....	177
12.3	Parameter Semantics	177
12.4	Interface Structure	179
12.5	Channel Specific Messages	179
12.5.1	The <i>channel_setup</i> message (SIMCOMP \Rightarrow MUXCONFIG).....	179
12.5.2	The <i>channel_status</i> message (SIMCOMP \Leftrightarrow MUXCONFIG).....	180
12.5.3	The <i>channel_test</i> message (SIMCOMP \Leftrightarrow MUXCONFIG)	181
12.5.4	The <i>channel_close</i> message (SIMCOMP \Leftrightarrow MUXCONFIG).....	182
12.5.5	The <i>channel_error</i> message (SIMCOMP \Leftrightarrow MUXCONFIG)	182
12.6	Transport Resource Mapping Messages.....	183
12.6.1	The <i>TS_resource_discover</i> message (SIMCOMP \Rightarrow MUXCONFIG).....	183
12.6.2	The <i>TS_resource_request</i> message (SIMCOMP \Rightarrow MUXCONFIG)	183
12.6.3	The <i>TS_resource_update</i> message (SIMCOMP \Leftrightarrow MUXCONFIG)	184
13	Timing and Play-out Issues	186
13.1	Timing issues.....	186
13.2	Delay Start.....	187
13.3	Play-out Issues.....	188
13.3.1	ECMs	188
13.3.2	EMMs and Private Data.....	188
13.4	Crypto-Period Realignment.....	188
Annex A (normative):	System Layering.....	190
A.1	Introduction	190
A.2	Physical Layer	190
A.3	Data Link Layer	190
A.4	Network Layer.....	190
A.5	Transport Layer	190
A.6	Session Layer	190
A.7	System Layering Overview/Communications Protocol stack	191
A.8	TCP or UDP Connection Establishment	192
Annex B (informative):	SCS Coexistence.....	193
B.1	Introduction	193
B.2	Example scenario	193
Annex C (informative):	Control word generation and testing	194
C.1	Introduction	194
C.2	Background	194
C.3	Generation	194
C.4	Control word randomness verification testing	195
C.4.1	1/0 bias	195
C.4.2	Autocorrelation.....	195
C.5	Testing locations	195

Annex D (informative):	Security Method for the SCS \Leftrightarrow ECMG Interface	196
D.1	Algorithm Selection	196
D.2	Control Word processing.....	197
D.3	Key Management	197
D.3.1	Key Generation/Distribution	197
D.3.2	Selection	198
D.3.3	Key Pointer Distribution	199
D.3.4	Fixed Key Mode	199
D.4	Encryption Function Toggling	200
Annex E (normative):	Summary of Requirements for C(P)SIG \Leftrightarrow (P)SIG interface	201
E.1	Head-end system requirements	201
E.2	CAS's C(P)SIG requirements	202
Annex F (informative):	C(P)SIG\Leftrightarrow(P)SIG Connection-oriented Configuration Example	204
F.1	Head-end processes and configuration data	204
F.2	CAS processes and configuration data.....	205
F.3	Channels and configuration data	205
F.4	Streams and configuration data	206
Annex G (normative):	Transition Timing for EIS \Leftrightarrow SCS	208
Annex H (normative):	Crypto-period duration management by the SCS	211
H.1	<i>Nominal_CP_duration</i> in ECMG \Leftrightarrow SCS protocol	211
H.2	Management of the <i>recommended_CP_duration</i> value	211
Annex I (normative):	Standard compliance	213
I.1	Overview	213
I.2	General compliance scheme for connection-based protocols.....	214
I.3	Functional difference between V2 and V3 in ECMG \Leftrightarrow SCS protocol.....	215
I.4	Functional differences between V2 and V3 in EMMG \Leftrightarrow PDG protocol	215
I.5	Functional differences between V2 and V3 in C(P)SIG \Leftrightarrow (P)SIG protocol.....	215
I.6	SIMF.....	215
I.6.1	Functional differences between V2 and V3.....	215
I.6.2	Recommendation for SIMF compliance.....	215
I.7	Functional differences between V3 and V4 in EIS \Leftrightarrow SCS protocol.....	216
I.8	Functional differences between V3 and V4 in (P)SIG \Leftrightarrow MUX protocol	216
I.9	Functional differences between V4 and V5 in ECMG \Leftrightarrow SCS and EMMG \Leftrightarrow MUX protocols	216
Annex J (informative):	Use of DVB ASI for the PSIG \Leftrightarrow MUX interface	217
Annex K (normative):	ASN.1 MIBs description.....	218
K.1	SIM MIB	218
K.2	SEM MIB	256
K.3	SLM MIB	272
Annex L (normative):	SIMCOMP\LeftrightarrowMUXCONFIG XML Schema Definition	284

Annex M (normative):	ACG ⇔ EIS XML Schema Definition	286
Annex N (normative):	ECMG ⇔ SCS and EMMG ⇔ MUX interfaces adaptations for use with IP Datacasting over DVB-H	289
N.1	Introduction	289
N.2	CP_CW_combination parameter in ECMG ⇔ SCS	289
N.3	Section_TSpkt_flag parameter in ECMG ⇔ SCS	290
N.4	Section_TSpkt_flag parameter in EMMG ⇔ MUX	290
N.5	IPsec Traffic Authentication Key Derivation	290
Annex O (informative):	Bibliography	292
History	293

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://webapp.etsi.org/IPR/home.asp>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECTrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

1 Scope

The present document of DVB-Simulcrypt addresses the requirements for interoperability between two or more conditional access systems at a head-end. It specifies the system architecture, timing relationships, messaging structures, extended interoperability and control.

The components within the system architecture represent functional units. The boundaries between physical units are not required to match the boundaries between functional units. It is possible that the SCS could be in the MUX or the SCS and MUX could be built independently. Neither architecture is mandated.

1.1 Common scrambling algorithm

The DVB-Simulcrypt group has looked at issues relating to the concepts of the common scrambling algorithm, within the DVB-Simulcrypt environment.

The DVB-Simulcrypt system is based on the concept of a shared scrambling and descrambling method. The group has looked at the possible constraints, which the DVB-Simulcrypt architecture might impose on the use of such a shared scrambling and descrambling method. No problems were noted.

1.2 Language

The word "shall" is used in a normative statement that can be verified and is mandatory. The word "should" is used in the context of a recommendation or a statement that cannot be verified or is not mandatory (it may be optional).

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
 - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
 - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at <http://docbox.etsi.org/Reference>.

For online referenced documents, information sufficient to identify and locate the source shall be provided. Preferably, the primary source of the referenced document should be cited, in order to ensure traceability. Furthermore, the reference should, as far as possible, remain valid for the expected life of the document. The reference shall include the method of access to the referenced document and the full network address, with the same punctuation and use of upper case and lower case letters.

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

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

- [1] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".
- [2] IETF RFC 1213 (1991): "Management Information Base for Network Management of TCP/IP-based internets: MIB-II".
- [3] ISO/IEC 13818-1 (2000): "Information technology; Generic coding of moving pictures and associated audio information: Systems".
- [4] ITU-T Recommendation X.731 (1992)/ISO/IEC 10164-2 (1993): "Information technology; Open Systems Interconnection; Systems Management: State Management Function".
- [5] ITU-T Recommendation X.733 (1992)/ISO/IEC 10164-4 (1992): "Information technology; Open Systems Interconnection; Systems Management: Alarm reporting function".
- [6] ITU-T Recommendation X.734 (1992)/ISO/IEC 10164-5 (1993): "Information technology; Open Systems Interconnection; Systems management: Event Report Management Function".
- [7] ITU-T Recommendation X.735 (1992)/ISO/IEC 10164-6 (1993): "Information technology; Open Systems Interconnection; Systems Management: Log control function".
- [8] IETF RFC 768 (1980): "User Datagram Protocol".
- [9] IETF RFC 791 (1981): "Internet Protocol".
- [10] IETF RFC 793 (1981): "Transmission Control Protocol".
- [11] IETF RFC 1901 (1996): "Introduction to Community-based SNMPv2".
- [12] IETF RFC 1902 (1996): "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [13] IETF RFC 1903 (1996): "Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [14] IETF RFC 1904 (1996): "Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [15] IETF RFC 1905 (1996): "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [16] IETF RFC 1906 (1996): "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [17] IETF RFC 1907 (1996): "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)".
- [18] IETF RFC 1908 (1996): "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework".
- [19] FIPS 46-2: "Data Encryption Standard (DES)" (supersedes FIPS 46-1).
- [20] CENELEC EN 50083-9: "Cable networks for television signals, sound signals and interactive services; Part 9: Interfaces for CATV/SMATV headends and similar professional equipment for DVB/MPEG-2 transport streams"; Annex B: "Asynchronous Serial Interface".
- [21] ETSI TS 101 197 (V1.2.1): "Digital Video Broadcasting (DVB); DVB SimulCrypt; Head-end architecture and synchronization".

- [22] ETSI TS 103 197 (V1.2.1): "Digital Video Broadcasting (DVB); Head-End Implementation of DVB Simulcrypt".
- [23] ISO/IEC 13818-2: "Information technology; Generic coding of moving pictures and associated audio information: Video".
- [24] ISO/IEC 13818-3: "Information technology; Generic coding of moving pictures and associated audio information - Part 3: Audio".
- [25] ETSI TS 103 197 (V1.3.1): "Digital Video Broadcasting (DVB); Head-End Implementation of DVB Simulcrypt".
- [26] ETSI TS 102 474: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Service Purchase and Protection".
- [27] ISMA: "Internet Streaming Media Alliance Implementation Specification - Encryption and Authentication".
- [28] IETF RFC 4301 (2005): "Security Architecture for the Internet Protocol".
- [29] IETF RFC 3711 (2004): "The Secure Real-time Transport Protocol (SRTP)".
- [30] IETF RFC 3566 (2003): "The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec".
- [31] IETF RFC 4434 (2006): "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)".
- [32] ETSI TS 103 197 (V1.4.1): "Digital Video Broadcasting (DVB); Head-end implementation of DVB SimulCrypt".

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

- [i.1] ETSI TR 101 154: "Digital Video Broadcasting (DVB); Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in satellite, cable and terrestrial broadcasting applications".
- [i.2] ETSI ETR 162: "Digital Video Broadcasting (DVB); Allocation of Service Information (SI) codes for DVB systems".
- [i.3] ETSI TR 101 211: "Digital Video Broadcasting (DVB); Guidelines on implementation and usage of Service Information (SI)".
- [i.4] ETSI ETR 289: "Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems".
- [i.5] ETSI TR 101 891: "Digital Video Broadcasting (DVB); Professional Interfaces: Guidelines for the implementation and usage of the DVB Asynchronous Serial Interface (ASI)".
- [i.6] ETSI TR 102 035: "Digital Video Broadcasting (DVB); Implementation Guidelines of the DVB Simulcrypt Standard".

3 Definitions and abbreviations

3.1 Definitions

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

access criteria: CA system specific information needed by the ECMG to build an ECM

Access Criteria Generator (ACG): See clause 4.2.16.

broadcaster (service provider): organization which assembles a sequence of events or services to be delivered to the viewer based upon a schedule

CA_subsystem_id: system which handles multiple connections to ECMGs with the same CA_system_id value

NOTE: The combination of CA_system_id and CA_subsystem_id is called *super_CAS_id*.

CA_system_id: See ETR 162 [i.2], table 3.

CA components: components brought by a CA provider for integration into a host head-end system

channel: application specific representation of an open TCP connection, allowing the association of application specific parameters with such a connection

NOTE: Channels correspond on a one to one basis to TCP connections.

client: software entity on a host making use of one or more resources offered by a server

commercial event information: subset of the information related to a single event provided by the Commercial world to a CA system to generate related Access Criteria

commercial world: service editors, operators, content providers and other bodies involved in production and commercial scheduling of TV programs

Conditional Access (CA) system: system to control subscriber access to broadcast services and events

Control Word (CW): data object used for scrambling

Control Word Generator (CWG): component which receives a CW request from the SCS and returns a CW

Crypto Period (CP): period when a particular Control Word is being used by the scrambler

Entitlement Control Message (ECM): private Conditional Access information, which carries the control word in a secure manner and private entitlement information

Entitlement Control Message Generator (ECMG): generator which produces the ECM messages but does not support ECM repetition

NOTE: See clause 4.2.3.

Entitlement Management Message (EMM): private Conditional Access information which, for example, specifies the authorization levels of subscribers or groups of subscribers for services or events

Entitlement Management Message Generator (EMMG): generator which produces the EMM messages and repeatedly plays them out at the appropriate times

NOTE: See clause 4.2.4.

event: scheduled change of the status or of the conditions to access a certain group of components (i.e. an SCG) broadcast in a transport stream generated by the head-end

event scheduling information: set of events for a defined period of time for one or more services controlled by the CAS

forbidden: indicates that the value shall never be used

generator: component producing data

host: computer system uniquely identified by its IP address, and as such addressable in a computer network

NOTE: It may take both client and server roles.

host head-end: system which is composed of those components required before a CA provider can be introduced into the head-end

MPEG-2: Refers to the standard ISO/IEC 13818, specifically:

- Systems coding is defined in ISO/IEC 13818-1 [3].
- Video coding is defined in ISO/IEC 13818-2 [23].
- Audio coding is defined in ISO/IEC 13818-3 [24].

multiplex: stream of all the digital data within a single physical channel carrying one or more services or events

Multiplexer (MUX): See clause 4.2.11.

Private Data Generator (PDG): See clause 4.2.5.

proprietary: fact that the interface will be specified by the head-end provider, or by the CA provider

NOTE: The interface can be commercially open but is not open within the present document. Its availability will be via commercial/technical agreement.

reserved: in the clause defining the coded bit stream, indicates that the value may be used in the future for ISO defined extensions, unless otherwise specified within the present document all "reserved" bits shall be set to "1"

reserved future use: in the clause defining the coded bit stream, indicates that the value may be used in the future for ETSI defined extensions, unless otherwise specified within the present document all "reserved_future_use" bits shall be set to "1".

resource: set of coherent functions, accessible through a server

NOTE: More than one resource can reside on a single host.

Scrambler (SCR): See clause 4.2.12.

Scrambling Control Group (SCG): data structure gathering together in one same logical set the list of A/V streams scrambled at the same time with the same control word and the list of ECMs that are going to be generated with the identifier of their CA system and with their respective Access Criteria

server: software entity exporting a resource

NOTE: More than one server may reside on a single host. A server is uniquely identified by an IP address and TCP port number.

service: sequence of events under the control of a broadcaster, which can be broadcast as part of a schedule

Service Information (SI): information that is transmitted in the transport stream to aid navigation and event selection

SI generator: See clause 4.2.10.

Simulcrypt Integrated Management Framework (SIMF): addresses the requirements for interoperability between management components of multiple conditional access systems (CASs) at a head-end

NOTE: See clause 7.

Simulcrypt Synchronizer (SCS): logical component that acquires Control Words, ECMs and synchronizes their play-out for all the Conditional Access Systems connected

stream: independent bi-directional data flow across a channel

NOTE: Multiple streams may flow on a single channel. Stream_ids (e.g. ECM_stream_id, data_stream_id, etc.) are used to tag messages belonging to a particular stream.

super_CAS_id: 32-bit identifier formed by the concatenation of the CA_system_id and the CA_subsystem_id

Transport Stream: data structure

NOTE: It is the basis of the ETSI Digital Video Broadcasting (DVB) standards, defined in ISO/IEC 13818-1 [3].

3.2 Abbreviations

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

AC	Access Criteria
ACG	Access Criteria Generator
AES	Advanced Encryption Standard
ASI	Asynchronous Serial Interface
ASN.1	Abstract Syntax Notation One
bslbf	bit string, left bit first
C(P)SIG	Custom PSI/SI Generator
CA	Conditional Access
CAS	Conditional Access System
CAT	Conditional Access Table
CaM	Carousel in the MUX
CIM	Common Information Model
CiP	Carousel in the (P)SIG
CORBA	Common Object Request Broker Architecture
CP	Crypto Period
CPSIG	Custom Program Specific Information Generator
CRC	Cyclical Redundancy Check
CSIG	Custom Service Information Generator
CW	Control Word
CWG	Control Word Generator
DVB	Digital Video Broadcasting
DVB-H	Digital Video Broadcasting for Handheld Terminals
EBU	European Broadcasting Union
ECM	Entitlement Control Message
ECMG	Entitlement Control Message Generator
EFD	Event Forwarding Discriminator
EGP	Exterior Gateway Protocol
EIS	Event Information Scheduler
EIT	Event Information Table
EMM	Entitlement Management Message
EMMG	Entitlement Management Message Generator
FDDI	Fibre Distributed Data Interface
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
Id	Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
IPDC	Internet Protocol Datacasting
IPsec	IP Security
ISMA	Internet Streaming Media Alliance
ISMAcryp	Internet Streaming Media Alliance encryption and authentication
ISO	International Organization for Standardization
JIDM	Joint Inter-Domain Management
JMAPI	Java Management API
KMM	Key Management Message
KSM	Key Stream Message
LAP	Logical Access Point
LSB	Least Significant Bit
MIB	Management Information Base
MISC	Metal Insulator Semiconductor Capacitor

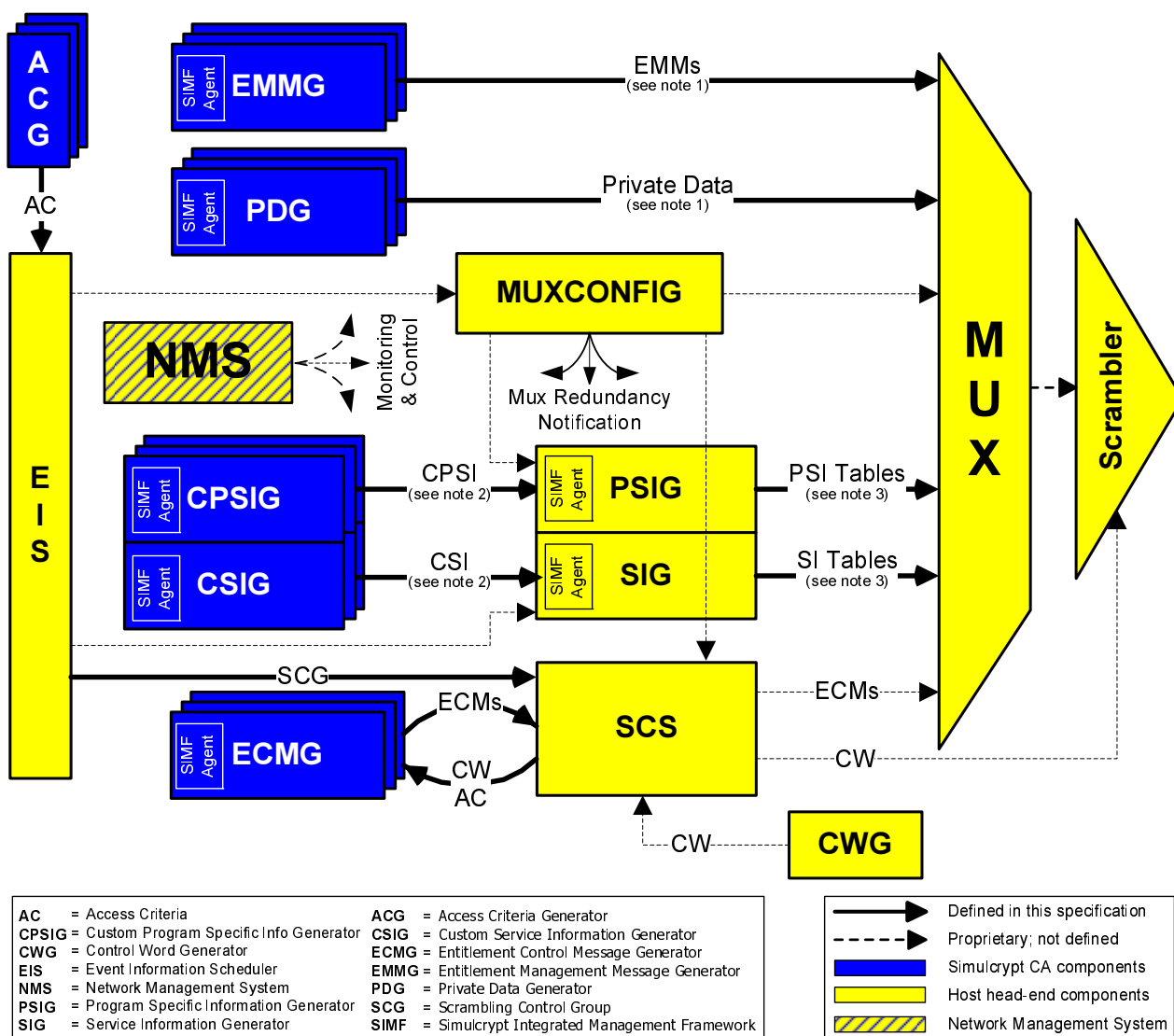
MJD	Modified Julian Date
MPEG	Moving Pictures Expert Group
MUX	MULTipleXer
MUXCONFIG	Multiplexer Configuration
NIT	Network Information Table
NM	Network Management
NMS	Network Management System
OID	Object IDentifier
ORB	Object Request Broker
ORP	Operations Reference Point
OSI	Open Systems Interconnection
PAT	Program Association Table
PD	Private Data
PDG	Private Data Generator
PID	Packet IDentifier
PMT	Program Map Table
PSI	Program Specific Information
PSIG	Program Specific Information Generator
SCG	Scrambling Control Group
SCR	DVB Compliant Scrambler
SCS	SimulCrypt Synchronizer
SDT	Service Description Table
SEM	Simulcrypt Events Module
SI	Service Information
SIG	Service Information Generator
SIM	Simulcrypt Identification Module
SIMCOMP	SIMulcrypt COMPonent
SIMF	Simulcrypt Integrated Management Framework
SLM	Simulcrypt Logs Module
SMI	Structure of Management Information
SMIB	Simulcrypt Management Information Base
SNMP	Simple Network Management Protocol
SPI	Synchronous Parallel Interface
S RTP	Secure Real-time Transport Protocol
SSI	Synchronous Serial Interface
STB	Set Top Box
tcimsbf	two's complement integer most significant bit first
TCP	Transport Control Protocol
TLV	Type, Length, Value
TMN	Telecommunications Management Network
TOT	Time Offset Table
TS	Transport Stream
UDP	User Datagram Protocol
uimsbf	unsigned integer most significant bit first
UTC	Universal Time, Co-ordinated
WBEM	Web-Based Enterprise Management
XML	eXtended Markup Language
TDT	Time and Date Table
LFSR	Linear Feedback Shift Register

4 Architecture

4.1 System architecture

Figure 1 shows the logical relationships between the components and which component-to-component interfaces are defined in the present document. Other components exist in a head-end, which are not illustrated in figure 1 (for example: SMS or Subscriber Management System, etc.).

The DVB-Simulcrypt system architecture illustrated above is divided into 3 areas. No assumption is made that different components belonging to the same area are to be provided by the same manufacturer.



NOTE 1: EMMG ⇄ MUX.

NOTE 2: C(P)SIG ⇄ (P)SIG.

NOTE 3: (P)SIG ⇄ MUX.

Figure 1: DVB Simulcrypt System Architecture

4.1.1 Host Head-end components

Host head-end components are those that will need to exist before Simulcrypt CA components can be introduced into a DVB-Simulcrypt head-end.

4.1.2 Simulcrypt CA components

Simulcrypt CA components are typically those, which are brought by a new CA provider to introduce his CA into a DVB-Simulcrypt head-end.

NOTE: The EMMGs, PDGs and Custom SI generators are not necessarily required in a DVB-Simulcrypt system.

4.1.3 Simulcrypt Integrated Management Framework (SIMF)

The components of multiple Conditional Access Systems (CASs) involved in a Simulcrypt architecture can be supported by a network management function existing in a head-end.

For ECMG, EMMG, PDG, C(P)SIG and (P)SIG, a Simulcrypt Integrated Management Framework (SIMF) is defined to address the requirements for interoperability between management components at a head-end. This framework does not address all of the issues relevant for a complete integrated Simulcrypt Management System, it specifies only the minimum set of components necessary to enable integration. SIMF is described in clause 7.

4.1.4 Multiplexer Redundancy

The Simulcrypt Architecture provides specific support for head-end implementations in which the multiplexer components are redundantly configured. An optional protocol is specified that allows Simulcrypt Components (SIMCOMPs) to discover the multiplexer redundancy topology and to request the current multiplexer connection information. This protocol also allows the Multiplexer Configuration (MUXCONFIG) components to asynchronously notify SIMCOMPs of redundancy fail-over events. The support for multiplexer redundancy is described in clause 12.

4.2 Description of Components

4.2.1 Event Information Scheduler (EIS)

In the DVB-Simulcrypt system architecture diagram (see clause 4.1), the EIS is the functional unit in charge of holding the entire schedule information, all the configurations and CA specific information required for the complete system. It is the overall database store for the whole head-end system. For instance, it is in charge of providing to the ECMGs (via the SCS) any information they need to generate their ECMs.

In reality this function might be distributed over several physical units, storage locations, and/or input terminals, and it may communicate with any other functional unit of the architecture diagram.

Concerning the CA provider components, the connections to the EIS and the data they carry will be agreed through the commercial arrangements made with the broadcaster. They are not defined in the present document.

4.2.2 Simulcrypt Synchronizer (SCS)

The role of the Simulcrypt Synchronizer is to:

- establish TCP connections with ECMGs and setup one channel per connection;
- setup streams within channels as needed and allocate ECM_stream_id values;
- get the control words from the CWG;
- supply the CWs to the relevant ECMGs on the relevant streams, as well as any CA specific information;
- acquire ECMs from the ECMGs;
- synchronize the ECMs with their associated Crypto periods according to channel parameters;
- submit these ECMs to the MUX and request their repetition according to the channel parameters;
- supply the CW to the scrambler for use in the specified Crypto Period.

4.2.3 ECM Generator (ECMG)

The ECMG shall receive CWs in a CW provision message as well as access criteria and shall reply with an ECM or an error message. The ECMG does not support ECM repetition.

4.2.4 EMM Generator (EMMG)

This component, supplied by the CA provider shall interface over a DVB-Simulcrypt specified interface to the MUX. The EMMG initiates connections to the MUX.

4.2.5 Private Data Generator (PDG)

This component is shown in the DVB-Simulcrypt System Architecture diagram to highlight the fact that the EMMG to MUX interface can be used for EMMs and other CA related private data. The PDG initiates connections to the MUX.

4.2.6 Service Information Generator (SIG)

This component is responsible for generating the SI for the system in the form of DVB SI tables (see EN 300 468 [1]). The SI Generator takes its primary data from the EIS and supplementary data from the Custom SI Generators supplied by the CA vendor. The interface between the EIS and the SIG is not specified in the present document.

In the context of SI generation, the generic term **(P)SIG** refers to a head-end process that serves as an SIG or a combined PSISIG.

If the head-end supports the $C(P)SIG \Leftrightarrow (P)SIG$ interface, it shall include at least one SIG. If the head-end supports the $(P)SIG \Leftrightarrow MUX$ interface, it shall include at least either one PSISIG or a pair (PSIG, SIG).

4.2.7 Program Specific Information Generator (PSIG)

This component is responsible for generating the PSI for the system in the form of MPEG-2 PSI tables (see ISO/IEC 13818-1 [3]). The PSI Generator takes its primary data from the MUXCONFIG and supplementary data from the Custom PSI Generators supplied by the CA vendor. The interface between the MUXCONFIG and the PSI Generator is not specified in the present document.

In the context of PSI generation, the generic term **(P)SIG** refers to a head-end process that serves as a PSIG or a combined PSISIG.

If the head-end supports the $C(P)SIG \Leftrightarrow (P)SIG$ interface, it shall include at least one PSIG. If the head-end supports the $(P)SIG \Leftrightarrow MUX$ interface, it shall include at least either one PSISIG or a pair (PSIG, SIG).

4.2.8 Custom Service Information Generator (CSIG)

This component is responsible for generating CAS-specific private data for insertion in selected DVB SI tables. It interfaces to the SI Generator. Each CAS may include one or more CSIGs.

In the context of private SI data generation, the generic term **C(P)SIG** refers to a head-end process that serves as a CSIG or a combined CPSISIG.

4.2.9 Custom Program Specific Information Generator (CPSIG)

This component is responsible for generating CAS-specific private data for insertion in selected MPEG-2 PSI tables and interfaces to the PSI Generator. Each CAS may include one or more CPSIGs.

In the context of private PSI data generation the generic term **C(P)SIG** refers to a head-end process that serves as a CPSIG or a combined CPSISIG.

4.2.10 Multiplexer Configuration (MUXCONFIG)

This component is responsible for configuring the MUX and providing a link to the PSI generator for PSI construction and play-out. The interfaces $MUXCONFIG \Leftrightarrow MUX$ and $MUXCONFIG \Leftrightarrow PSIG$ are not defined by the present document.

4.2.11 Multiplexer (MUX)

The role of this head-end component is to perform time multiplexing of input data, and to output an MPEG-2 transport stream. The input data can be transport packets, MPEG sections or raw data. The exact functionality of a MUX is implementer specific. For the purpose of the present document, the MUX shall be able to communicate with the SCS, the (P)SIG and to accept connections with EMMGs with the interface defined.

4.2.12 Scrambler (SCR)

This component is responsible for scrambling data in the MPEG2 Transport Stream. The exact functionality of the Scrambler is implementer specific. For the purpose of the present document, the Scrambler shall be able to receive Control Words from the SCS.

4.2.13 Control Word Generator (CWG)

This component is responsible for generating control words used in scrambler initialization stream. The exact functionality of the Scrambler is implementer specific. For the purpose of the present document, the Control Word Generator shall be able to provide the SCS with control words.

4.2.14 Network Management System (NMS)

This component is responsible for monitoring and control of SIMF agents. The exact nature of this function depends on the type of host component the agent is situated in, i.e. ECMG, EMMG, PDG, etc., and the type of management function the NMS component is performing, i.e. fault, configuration, accounting, performance and security management.

4.2.15 SIMF agent

This component supports network management protocol transactions on the Simulcrypt Management Information Base (SMIB), which it implements. It instruments the SMIB with monitoring and control functionality of the host component, i.e. ECMG, EMMG, PDG, etc.

4.2.16 Access Criteria Generator (ACG)

This component, supplied by the CA provider is responsible for generating CAS related information, e.g. access criteria, related to each event managed by the EIS when this event involves its associated CAS.

4.3 Description of interfaces

4.3.1 ECMG \Leftrightarrow SCS

The interface allowing a CAS to provide a SCS with ECMs under the control of this SCS. This interface is mandatory and shall be implemented as described in clause 5.

4.3.2 EMMG \Leftrightarrow MUX

The interface allowing a CAS to provide a MUX with EMM under the control of the CAS. This interface is mandatory and shall be implemented as described in clause 6.

4.3.3 PDG \Leftrightarrow MUX

The interface is identical to the EMMG \Leftrightarrow MUX interface, and may be used for providing CA related private data to the MUX for insertion in the transport stream. This interface is optional; if implemented it shall be implemented as per the EMMG \Leftrightarrow MUX interface described in clause 6.

4.3.4 Custom (P)SI Generator \Leftrightarrow (P)SI Generator

The interface allowing a CAS to provide a (P)SIG with private data descriptors for the head-end to insert in (P)SI tables. This interface is mandatory and shall be implemented as described in clause 8.

4.3.5 EIS \Leftrightarrow SIG

Proprietary, not defined by the present document.

4.3.6 (P)SI Generator \Leftrightarrow MUX

The interface used to provide the MUX with the adequate PSI/SI tables of the transport stream it generates. This interface is mandatory and shall be implemented as described in clause 9.

4.3.7 EIS \Leftrightarrow MUXCONFIG

Proprietary, not defined by the present document.

4.3.8 MUXCONFIG \Leftrightarrow PSIG

Proprietary, not defined by the present document.

4.3.9 MUXCONFIG \Leftrightarrow SCS

Proprietary, not defined by the present document.

4.3.10 MUX \Leftrightarrow SCR

Proprietary, not defined by the present document.

4.3.11 SCR onward

Proprietary, not defined by the present document.

4.3.12 SCS \Leftrightarrow MUX

Proprietary, not defined by the present document.

4.3.13 SCS \Leftrightarrow SCR

Proprietary, not defined by the present document.

4.3.14 SCS \Leftrightarrow CWG

Proprietary, not defined by the present document.

4.3.15 EIS \Leftrightarrow SCS

The interface providing the SCS with the Scrambling Control Groups. An SCG actually identifies the service(s) and/or elementary stream(s) in one particular Transport Stream:

- which are to be scrambled at each moment using a common Control Word (CW) key;
- which are associated to the same ECM streams.

This interface is mandatory and shall be implemented as described in clause 10.

4.3.16 ACG \Leftrightarrow EIS

This interface allows different CA providers in a Simulcrypt configuration to submit proprietary CAS information to the EIS on request. The ACG takes on the responsibility of mapping this "editorial" CA information into access criteria and other proprietary CAS related information so that the EIS, SCS, and PSIG may remain CA vendor neutral.

This interface also supports a mechanism to allow the ACG to notify the EIS that AC information previously transmitted to the EIS needs to be updated for CAS-proprietary reasons. In this case, a referencing mechanism is defined to allow the ACG to identify the AC value that needs to be updated.

This interface is mandatory and shall be implemented as described in clause 11.

4.3.17 NMS Component \Leftrightarrow SIMF Agent

This interface allows a network management function existing in a head-end to support Simulcrypt CA components. It is defined by the Simulcrypt MIB and the NM protocol used in the system.

This interface is optional; if used it shall be implemented as described in clause 7.

4.3.18 SIMCOMP \Leftrightarrow MUXCONFIG

The interface between any Simulcrypt component and a Multiplex Configuration component that provides support for multiplexer redundancy in the head-end. This interface supports both hot and cold redundancy in an N:M configuration and allows SIMCOMPs to request connectivity information from MUXCONFIGs, and for MUXCONFIGs to signal redundancy switch-over events to SIMCOMPs.

This interface is optional; if used it shall be implemented as described in clause 12.

4.3.19 Mandatory or optional characteristics of Simulcrypt interfaces

Clauses 4.3.1 to 4.3.18 give the global mandatory or optional characteristic of each interface described in the present document. Table 1a sums up these characteristics.

Table 1a: Mandatory or optional characteristics of the Simulcrypt interfaces

Interfaces	Global Characteristic	Particular points	See clauses
ECMG \Leftrightarrow SCS	mandatory	only TCP based implementation Security of control words: <ul style="list-style-type: none"> CW security shall be supported; the means or methods to support this security are chosen by commercial agreement; one of them is the CW encryption in the protocol; CW encryption in the protocol is optional if CWs are encrypted in the protocol, the method given in annex D is recommended. 	5.1.7
EMMG \Leftrightarrow MUX	mandatory	a real implementation is chosen by the head-end operator among: <ul style="list-style-type: none"> TCP based implementation for data provision and control; UDP based implementation for data provision and TCP based implementation for control; UDP based implementation for data provision and SIMF based implementation for control. 	6.1
C(P)SIG \Leftrightarrow (P)SIG	mandatory	according to the same application protocol model, a real implementation is defined by commercial agreement among: <ul style="list-style-type: none"> TCP based implementation; SIMF based implementation. 	8.2

Interfaces	Global Characteristic	Particular points	See clauses
NMS \Leftrightarrow SIMF Agent	optional	the Simulcrypt Network Management function implementation is optional and is defined by the head-end operator among: <ul style="list-style-type: none"> • SNMP v2 for agents and manager; this implementation is fully defined in the present document; • SNMP v2 for agents and CORBA for manager; in this case only the SNMP v2 compliant SMIB is described in the present document. 	7.1
(P)SIG \Leftrightarrow MUX	mandatory	a real implementation shall include <ul style="list-style-type: none"> • PSI/SI table carouselling performed by the PSIG; • PSI/SI table carouselling performed by the MUX. In case of PSI/SI table carouselling performed by the PSIG, a real implementation is defined by a commercial agreement among: <ul style="list-style-type: none"> • TCP based implementation for data and control. ASI based implementation for data and TCP based implementation for control	9.1
EIS \Leftrightarrow SCS	mandatory	only TCP based implementation	10.1
ACG \Leftrightarrow EIS	mandatory	only TCP based implementation	11.1
SIMCOMP \Leftrightarrow MUXCONFIG	optional	only TCP based implementation	12.1

4.4 Protocol types

4.4.1 Connection-oriented TLV protocols

For the connection-oriented protocols defined in the present document, the messages shall have the following generic structure.

Table 1b: Message-type values for command/response-based protocols

```

generic_message
{
    protocol_version      1 byte
    message_type          2 bytes
    message_length        2 bytes
    for (i=0; i < n; i++)
    {
        parameter_type    2 bytes
        parameter_length  2 bytes
        parameter_value   <parameter_length> bytes
    }
}

```

NOTE 1: For parameters with a size two or more bytes the first byte to be transmitted will be the most significant byte.

NOTE 2: Parameters do not need to be ordered within the generic message.

NOTE 3: The Boolean value "TRUE" is represented with the byte value 0x01, while the Boolean value "FALSE" is represented with the byte value 0x00.

protocol_version: An 8 bit field identifying the protocol version. It shall have the value according to table 2.

Table 2: Values for protocol_version parameter

Interfaces	Version
ECMG ⇔ SCS	0x03
EMMG ⇔ MUX	0x03
C(P)SIG ⇔ (P)SIG	0x03
NMS ⇔ SIMF Agent	0x03
(P)SIG ⇔ MUX	0x04
EIS ⇔ SCS	0x04

NOTE 4: Compliance between protocol versions is described in annex I.

NOTE 5: Protocol version 0x05 on the ECMG ⇔ SCS and EMMG ⇔ MUX interface is specifically for use with "IP Datacasting over DVB-H" and is described in annex N.

message_type: 16-bit field identifying the type of the message. The list of message type values is defined in table 3. Unknown message types shall be ignored by the receiving entity.

message_length: 16-bit field specifies the number of bytes in the message immediately following the message_length field.

parameter_type: 16-bit field specifies the type of the following parameter. The list of parameter type values is defined in the interface specific clauses of the present document. Unknown parameters shall be ignored by the receiving entity. The data associated with that parameter will be discarded and the remaining message processed.

parameter_length: 16-bit field specifies the number of bytes of the following parameter_value field.

parameter_value: variable length field specifies the actual value of the parameter. Its semantics is specific to the parameter type value.

Table 3: Message-type values for command/response-based protocols

Relevant interface	Message_type value	Message type
DVB reserved	0x0000	DVB reserved
ECMG ⇔ SCS	0x0001	channel_setup
	0x0002	channel_test
	0x0003	channel_status
	0x0004	channel_close
	0x0005	channel_error
DVB reserved	0x0006 to 0x0010	DVB reserved
EMMG ⇔ MUX	0x0011	channel_setup
	0x0012	channel_test
	0x0013	channel_status
	0x0014	channel_close
	0x0015	channel_error
DVB reserved	0x0016 to 0x0100	DVB reserved
ECMG ⇔ SCS	0x0101	stream_setup
	0x0102	stream_test
	0x0103	stream_status
	0x0104	stream_close_request
	0x0105	stream_close_response
	0x0106	stream_error
DVB reserved	0x107 to 0x110	DVB reserved
EMMG ⇔ MUX	0x0111	stream_setup
	0x0112	stream_test
	0x0113	stream_status
	0x0114	stream_close_request
	0x0115	stream_close_response
	0x0116	stream_error
	0x0117	stream_BW_request
	0x0118	stream_BW_allocation
DVB reserved	0x0119 to 0x0200	DVB reserved
ECMG ⇔ SCS	0x0201	CW_provision
	0x0202	ECM_response
DVB reserved	0x0203 to 0x0210	DVB reserved

Relevant interface	Message_type value	Message type
EMMG ⇔ MUX	0x0211	data_provision
DVB reserved	0x0212 to 0x0300	DVB reserved
C(P)SIG ⇔ (P)SIG	0x0301 0x0302 0x0303 0x0304 0x0305	channel_setup channel_status channel_test channel_close channel_error
DVB reserved	0x0306 to 0x0310	DVB reserved
C(P)SIG ⇔ (P)SIG	0x0311 0x0312 0x0313 0x0314 0x0315 0x0316 0x0317 0x0318 0x0319 0x031A 0x031B 0x031C 0x031D 0x031E 0x031F 0x0320 0x0321	stream_setup stream_status stream_test stream_close stream_close_request stream_close_response stream_error stream_service_change stream_trigger_enable_request stream_trigger_enable_response trigger table_request table_response descriptor_insert_request descriptor_insert_response PID_provision_request PID_provision_response
DVB reserved	0x0322 to 0x0400	DVB reserved
EIS ⇔ SCS	0x0401 0x0402 0x0403 0x0404 0x0405 0x0406 0x0408 0x0409 0x040A 0x040B 0x040C 0x040D	channel_set-up channel_test channel_status channel_close channel_error channel_reset SCG_provision SCG_test SCG_status SCG_error SCG_list_request SCG_list_response
DVB reserved	0x040E to 0x410	DVB reserved
(P)SIG ⇔ MUX	0x0411 0x0412 0x0413 0x0414 0x0415 0x0416 to 0x0420 0x0421 0x0422 0x0423 0x0424 0x0425 0x0426 0x0427 to 0x0430	channel_set_up channel_test channel_status channel_close channel_error Reserved stream_setup stream_test stream_status stream_close_request stream_close_response stream_error DVB Reserved
(Carousel in the MUX – CiM)	0x0431 0x0432 0x0433 to 0x0440	CiM_stream_section_provision CiM_channel_reset DVB Reserved
(Carousel in the (P)SIG – CiP)	0x0441 0x0442 0x0443	CiP_stream_BW_request CiP_stream_BW_allocation CiP_stream_data_provision
DVB reserved	0x0444 to 0x7FFF	DVB reserved
User defined	0x8000 to 0xFFFF	User defined

4.4.2 Connection oriented XML protocols

All messages exchanged on XML-based Simulcrypt protocols share a common general format as detailed in this clause. The messages are encoded using XML over a TCP connection.

All messages share a common structure that wraps the data for the specific message being sent. Each message follows a standard format starting with the length of the data followed by the actual data. All of the messages have an XML document as the actual data payload.

UTF-8 encoding shall be specified for the complete XML data payload.

Table 4: Message Structure for XML-based Protocols

Syntax	Bytes	Type
<pre>xml_message() { message_length message_data }</pre>	4 *	uimsbf Varies

message_length: The size of the message being sent in bytes (total from *message_length* through the end of data ()).

message_data: An XML document of varying size.

4.4.3 SIMF-based protocols

All Simulcrypt CAS/Head-end interface specifications such as ECMG \Leftrightarrow SCS, EMMG \Leftrightarrow MUX, PDG \Leftrightarrow MUX, C(P)SIG \Leftrightarrow (P)SIG, (P)SIG \Leftrightarrow MUX and EIS \Leftrightarrow SCS are based on a connection-oriented communications paradigm (i.e. channel/streams). Alternatively, some of these interfaces can be implemented using a transaction-based communications paradigm using the SIMF. In the current Simulcrypt specification this alternative is only available:

- for the C(P)SIG \Leftrightarrow (P)SIG interface;
- for the control of UDP-based protocol on EMMG/PDG \Leftrightarrow MUX interface.

5 ECMG \Leftrightarrow SCS interface

5.1 Interface principles

5.1.1 Channel and Stream specific messages

This interface shall carry the following channel messages defined further in clause 5.4:

- *Channel_setup*.
- *Channel_test*.
- *Channel_status*.
- *Channel_close*.
- *Channel_error*.

and the following stream messages defined further in clause 5.5:

- *Stream_setup*.
- *Stream_test*.
- *Stream_status*.

- *Stream_close_request*.
- *Stream_close_response*.
- *Stream_error*.
- *CW_provision*.
- *ECM_response*.

For this interface, the SCS is the client and the ECMG is the server. The SCS has a prior knowledge of the mapping between Super_CAS_ids and the IP addresses and port numbers of the ECMGs. When a new ECM stream is requested by the EIS for a given Super_CAS_id value, the SCS will open a new stream with the appropriate ECMG. This might require the opening of a new channel (which involves the opening of a new TCP connection).

When a new ECM stream is created in a transport stream, a new ECM_id shall be assigned to it by the head-end, according to the operational context (ECM stream creation or ECMG replacement). The value of the ECM_id parameter remains unmodified as long as the ECM stream exists. The combination {« ECM » type + Super_CAS_id + ECM_id} identifies uniquely this new ECM stream in the whole system.

NOTE: There can be several ECMGs associated with the same Super_CAS_id value (e.g. for performance or redundancy reasons). In such a case the SCS should be able to choose with which ECMG the connection will be opened, based on either a redundancy policy or resource available.

5.1.2 Channel establishment

There is always one (and only one) channel per TCP connection. Once the TCP connection is established, the SCS sends a channel_setup message to the ECMG. In case of success the ECMG replies by sending back a channel_status message.

In case of a rejection or a failure during channel setup the ECMG replies with a channel_error message. This means that the channel has not been opened by the ECMG and the SCS shall close the TCP connection.

5.1.3 Stream establishment

The SCS sends a stream_setup message to the ECMG. In case of success the ECMG replies by sending back a stream_status message. In case of a rejection or a failure the ECMG replies with a stream_error message.

Once the connection, channel and stream have been correctly established the ECM will be transferred. It can be transferred as sections or as TS packets. The ECMG indicates at channel setup which kind of data object will be used.

Once the connection, channel and stream have been correctly established, ECMs will be transferred to the SCS as a response to the CW_provision message.

5.1.4 Stream closure

Stream closure is always initiated by the SCS. This can occur when an ECM stream is no longer needed or in the case of an error. This is done by means of a stream_close_request message. A stream_close_response message indicates the stream has been closed.

5.1.5 Channel closure

Channel closure can occur when the channel is no longer needed or in case of error (detected by SCS or reported by ECMG). This is done by means of a channel_close message sent by the SCS. Subsequently, the connection shall be closed by both sides.

5.1.6 Channel/Stream testing and status

At any moment either component can send a channel_test/stream_test message to check the integrity of a channel/stream. In response to this message the receiving component shall reply with either a channel/stream status message or a channel/stream error message.

5.1.7 Unexpected communication loss

Both SCS and ECMG shall be able to handle an unexpected communication loss (either on the connection, channel or stream level).

Each component, when suspecting a possible communication loss (e.g. a 10 second silent period), should check the communication status by sending a test message and expecting to receive a status message. If the status message is not received in a given time (implementation specific) the communication path should be re-established.

5.1.8 Handling data inconsistencies

If the ECMG detects an inconsistency it shall send an error message to the SCS. If the SCS receives such a message or detects an inconsistency it may close the connection. The SCS (as the client) will then (re-)establish the connection, channel and (if applicable) streams.

The occurrence of a user defined or unknown parameter_type or message_type shall not be considered as an inconsistency.

5.2 Parameter_type values

Table 5: ECMG protocol parameter_type values

Parameter_type Value	Parameter type	Type/units	Length (bytes)
0x0000	DVB Reserved	-	-
0x0001	Super_CAS_id	uimbsf	4
0x0002	section_TSpkt_flag	uimbsf	1
0x0003	delay_start	tcimbsf/ms	2
0x0004	delay_stop	tcimbsf/ms	2
0x0005	transition_delay_start	tcimbsf/ms	2
0x0006	transition_delay_stop	tcimbsf/ms	2
0x0007	ECM_rep_period	uimbsf/ms	2
0x0008	max_streams	uimbsf	2
0x0009	min_CP_duration	uimbsf/n x 100ms	2
0x000A	lead_CW	uimbsf	1
0x000B	CW_per_msg	uimbsf	1
0x000C	max_comp_time	uimbsf/ms	2
0x000D	access_criteria	user defined	variable
0x000E	ECM_channel_id	uimbsf	2
0x000F	ECM_stream_id	uimbsf	2
0x0010	nominal_CP_duration	uimbsf/n x 100ms	2
0x0011	access_criteria_transfer_mode	Boolean	1
0x0012	CP_number	uimbsf	2
0x0013	CP_duration	uimbsf/n x 100ms	2
0x0014	CP_CW_Combination	---	variable
	CP	uimbsf	2
	CW	uimbsf	variable
0x0015	ECM_datagram	user defined	variable
0x0016	AC_delay_start	tcimbsf/ms	2
0x0017	AC_delay_stop	tcimbsf/ms	2
0x0018	CW_encryption	user defined	variable
0x0019	ECM_id	uimbsf	2
0x001A to 0x6FFF	DVB reserved	-	-
0x7000	Error_status	see clause 5.6	2
0x7001	Error_information	user defined	variable
0x7002 to 0x7FFF	DVB reserved	-	-
0x8000 to 0xFFFF	User defined	-	-

5.3 Parameter semantics

AC_delay_start: this parameter shall be used in place of the delay start parameter for the first Crypto period following a change in AC.

AC_delay_stop: this parameter shall be used in place of the delay stop parameter for the last Crypto period preceding a change in AC.

access_criteria: this parameter contains CA system specific information of undefined length and format, needed by the ECMG to build an ECM. It can be, for example, a pointer to an access criterion in an ECMG database, or a list of relevant access criteria items in an encapsulated TLV format. This parameter contains the information related to the CP indicated in the CW_provision message. The presence and contents of the access criteria parameter are the result of CA system supplier requirements.

access_criteria_transfer_mode: this 1-byte parameter is a flag. If it equals 0, it indicates that the access_criteria parameter is required in the CW_provision message only when the contents of this parameter change. If it equals 1, it indicates that the ECMG requires the access_criteria parameter be present in each CW_provision message.

CP_CW_combination: this parameter is the concatenation of the Crypto period number the control word is attached to and the control word itself. The parity (odd or even) of the Crypto Period number is equal to the parity of the corresponding control word (see ETR 289 [i.4]). This parameter is typically 10 byte long.

CP_duration: this parameter indicates the actual duration of a particular Crypto period for a particular stream when it differs from the nominal_CP_duration value (see definition below).

CP_number: an identifier to a Crypto period. This parameter indicates the Crypto period number a message is attached to. This is relevant for the following messages:

- CW_provision; and
- ECM_response.

CW_encryption: this parameter enables encrypting of control words over the SCS \Leftrightarrow ECMG interface. If the parameter is included in the CW_provision message, control word scrambling is invoked; if omitted, CWs are being issued in the clear. This parameter may include sub-parameters according to the used encrypting method. It may be used by the CW security method described in annex D or by an equivalent method.

CW_per_msg: the number of control words needed by the ECMG per control word provision message. If this value is "y" and lead_CW is "x", each control word provision message attached to Crypto period "n" will contain all control words from period $(n+1+x-y)$ to period $(n+x)$. Control words are carried with their Crypto period number by means of the CP_CW_combination parameter. In most existing CA systems CW_per_msg is 1 or 2. See also lead_CW.

For example, if an ECMG requires the current and next control word to generate an ECM, it shall by definition specify at least one lead_CW. However, since it may buffer its own control words, it can set CW_per_msg to one. By doing this, it always receives the control word for the *next* Crypto Period and accessing the control word for the current Crypto Period from memory (a previous provision message). Alternatively, it may specify 2 CW_per_msg and have both control words available at ECM generation time. This eliminates the need for ECMG buffering and can be advantageous for a hot backup to take over, since each provision message includes all control words required.

An SCS shall minimally support CW_per_msg values 1 and 2.

delay_start: this signed integer represents the amount of time between the start of a Crypto Period, and the start of the broadcasting of the ECM attached to this period. If it is positive, it means that the ECM shall be delayed with respect to the start of the Crypto Period. If negative, it means that the ECM shall be broadcast ahead of this time. This parameter is communicated by the ECMG to the SCS during the channel setup.

delay_stop: this signed integer represents the amount of time between the end of a Crypto Period, and the end of the broadcasting of the ECM attached to this period. If it is positive, it means that the end of the ECM broadcast shall be delayed with respect to the end of the Crypto Period. If negative, it means that the ECM broadcast shall be ended ahead of time. This parameter is communicated by the ECMG to the SCS during the channel setup.

ECM_channel_id: the ECM_channel_id is allocated by the SCS and uniquely identifies an ECM channel across all connected ECMGs.

ECM_id: the ECM_id is allocated by the head-end and uniquely identifies an ECM stream for a Super_CAS_id. The combination of the « ECM » type, the Super_CAS_id and the ECM_id identifies uniquely an ECM stream in the whole system. The unique identifier principle is described in clause 8.2.7.

ECM_datagram: the actual ECM message to be passed by the SCS to the MUX. According to the value of section_TSpkt_flag, the ECM_datagram can be either a series of transport packets (of 188 byte length), an MPEG-2 section, or an arbitrary length datagram in accordance with the IP Datacast over DVB-H Service Purchase and Protection specification [26] (refer to annex N). The ECM datagram can have a zero length, meaning that there is no ECM to be broadcast for the crypto period. The ECM datagram shall comply with ETR 289 [i.4].

ECM_rep_period: this integer represents the period in milliseconds for the repetition of data (e.g. ECMs).

ECM_stream_id: this identifier uniquely identifies an ECM stream within a channel. It is allocated by the SCS prior to stream setup.

error_status: see clause 5.6.

error_information: this optional parameter contains user defined data completing the information provided by error_status. It can be an ASCII text or the parameter ID value of a faulty parameter for example.

lead_CW: the number of control words required in advance to build an ECM. If this value is "x" the ECMG requires control words up to Crypto Period number "n+x" to build the ECM attached to Crypto period "n". In most existing CA systems lead_CW is 0 or 1. See also CW_per_msg.

For example, if the ECMG requires the current and next control word to generate an ECM, lead_CW would be 1. In other words, it defines the most future control word required for ECM generation.

An SCS shall minimally support lead_CW values 0 and 1.

max_comp_time: this parameter is communicated by the ECMG to the SCS during channel setup. It is the worst case time needed by an ECMG to compute an ECM when all the streams in a channel are being used. This time is typically used by the SCS to decide when to time-out on the ECM_response message. This value shall be lower than the min_CP_duration parameter of the same channel_status message.

max_streams: maximum number of simultaneous opened streams supported by an ECMG on a channel. This parameter is communicated from the ECMG to the SCS during the channel setup. A value of 0 means that this maximum is not known.

min_CP_duration: this parameter is communicated at channel setup by the ECMG to the SCS to indicate the minimum supported amount of time a control word shall be active before it can be changed. This value shall be greater than the max_comp_time parameter of the same channel_status message.

nominal_CP_duration: this parameter indicates the nominal duration of Crypto periods for the particular stream. It means that all the Crypto periods related to this stream will have this duration, except for the purpose of event alignments and error handling. This parameter is set up by the SCS (see annex H).

In addition, the nominal Crypto period duration (Nominal_CP_duration) and the actual Crypto-period (CP_duration) shall in any case be greater than or equal to:

- all the min_CP_duration specified by the ECMGs during Channel_set-up;
- all the max_comp_time values specified by the ECMGs during channel set-up, plus typical network latencies.

section_TSpkt_flag: this parameter defines the format of the ECM carried on this interface:

- **0x00:** the ECMs carried on the interface are in MPEG-2 section format;
- **0x01:** the ECMs carried on the interface are in MPEG-2 transport stream packet format, all TS packets shall be 188 byte long, any other payload length being considered as an error; it is the head-end's responsibility to fill the PID field in TS packet header;
- **0x02:** the ECMs carried on this interface are arbitrary length ECMs/KSMs in accordance with the IP Datacast Service Purchase and Protection specification [26] (refer to annex N);
- other values: DVB reserved.

Super_CAS_id: the Super_CAS_id is a 32-bit identifier formed by the concatenation of the CA_system_id (16 bit) and the CA_subsystem_id (16 bit). It shall identify uniquely a (set of) ECMG(s) for a given SCS, see clause 4.3.1. The CA_subsystem_id is defined by the user, it is private.

transition_delay_start: this parameter shall be used in place of the delay start parameter for the first crypto period following a clear to scrambled transition.

transition_delay_stop: this parameter shall be used in place of the delay stop parameter for the last crypto period preceding a scrambled to clear transition.

5.4 Channel specific Messages

5.4.1 Channel_setup message: ECMG \Leftarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
Super_CAS_id	1

The channel_setup message (message_type = 0x0001) is sent by the SCS to setup a channel once the TCP connection has been established, as described in clause 5.1.2. It shall contain the Super_CAS_id parameter, to indicate to the ECMG to which CA system and subsystem the channel is intended (indeed, there could be several Super_CAS_ids handled by a single ECMG host).

5.4.2 Channel_test message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1

The channel_test message (message_type = 0x0002) can be sent at any moment by either side to check:

- the channel is in an error free situation;
- the TCP connection is still alive.

The peer shall reply with a channel_status message if the channel is free of errors, or a channel_error message if errors occurred.

5.4.3 Channel_status message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
section_TSpkt_flag	1
AC_delay_start	0/1
AC_delay_stop	0/1
delay_start	1
delay_stop	1
transition_delay_start	0/1
transition_delay_stop	0/1
ECM_rep_period	1
max_streams	1
min_CP_duration	1
lead_CW	1
CW_per_msg	1
max_comp_time	1

The channel_status message (message_type = 0x0003) is a reply to the channel_setup message or the channel_test message (see clauses 5.1.2 and 5.1.6).

When the message is a response to a setup, the values of the parameters are those requested by the ECMG. All these parameter values will be valid during the whole lifetime of the channel, for all the streams running on it.

When the message is a response to a test, the values of the parameters shall be those currently valid in the channel.

5.4.4 Channel_close message: ECMG \Leftarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1

The channel_close message (message_type = 0x0004) is sent by the SCS to indicate the channel is to be closed.

5.4.5 Channel_error message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
error_status	1 to n
error_information	0 to n

A channel_error message (message type = 0x0005) is sent by the recipient of a channel_test message or by the ECMG at any time to indicate that an unrecoverable channel level error occurred. A table of possible error conditions can be found in clause 5.6.

5.5 Stream specific messages

5.5.1 Stream_setup message: ECMG \Leftarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1
ECM_id	1
nominal_CP_duration	1

The stream_setup message (message type = 0x0101) is sent by the SCS to setup a stream once the channel has been established, as described in clause 5.1.3.

5.5.2 Stream_test message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1

The stream_test message (message_type = 0x0102) is used to request a stream_status message for the given ECM_channel_id and ECM_stream_id. The stream_test message can be sent at any moment by either entity. The peer shall reply with a stream_status message if the stream is free of errors, or a stream_error message if errors occurred.

5.5.3 Stream_status message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1
ECM_id	1
access_criteria_transfer_mode	1

The stream_status message (message_type = 0x0103) is a reply to the stream_setup message or the stream_test message.

When the message is a response to a setup, the value of the access_criteria_transfer_mode parameter is the one requested by the ECMG.

When the message is a response to a test, the values of the parameters shall be those currently valid in the stream.

5.5.4 Stream_close_request message: ECMG \Leftarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1

The ECM_stream_id is sent by the SCS in the stream_close_request message (message type = 0x0104) to indicate which of the streams in a channel is due for closure.

5.5.5 Stream_close_response message: ECMG \Rightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1

The ECM_stream_id is sent by the ECMG in the stream_close_response message (message type = 0x0105) to indicate which of the streams in a channel is closing.

5.5.6 Stream_error message: ECMG \Leftrightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1
error_status	1 to n
error_information	0 to n

A stream_error message (message type = 0x0106) is sent by the recipient of a stream_test message or by the ECMG at any time to indicate that an unrecoverable stream level error occurred. A table of possible error conditions can be found in clause 5.6.

5.5.7 CW_provision message: ECMG \Leftarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1
CP_number	1
CW_encryption	0 to 1
CP_CW_combination	CW_per_msg
CP_duration	0 to 1
access_criteria	0 to 1

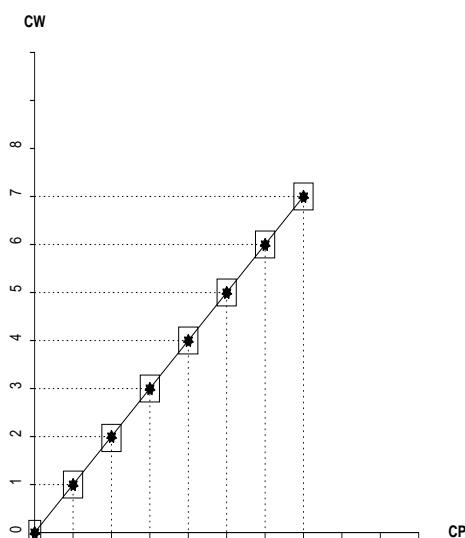
CW_provision message (message_type 0x201) is sent by the SCS to the ECMG and serves as a request to compute an ECM. The value of the CP_number parameter is the Crypto period number of the requested ECM. The control words are carried by this message with their associated Crypto period numbers in the CP_CW_combination parameter, according to the value of lead_CW and CW_per_msg as defined during the channel setup. For instance, if lead_CW = 1 and CW_per_msg = 2, the CW_provision message for Crypto period N shall contain control words for Crypto periods N and N+1.

The SCS is not allowed to send a CW_provision message before having received the ECM_response message for the previous Crypto periods, except if there has been a time-out expiration, or an error message (in which case the way this error is handled is left to the discretion of the SCS manufacturer).

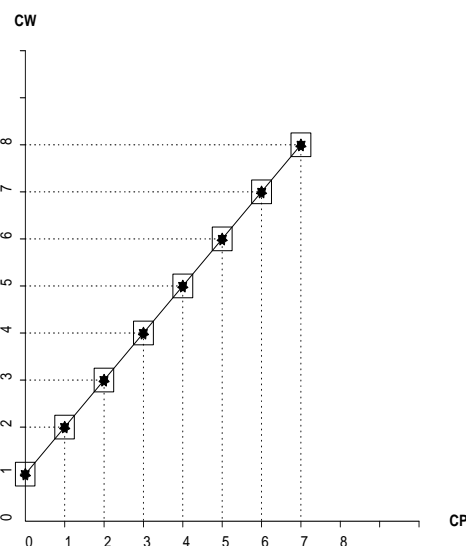
The specific CWs that are passed in the CP_CW_combination to the ECMG via the CW_provision message are derived from the values of lead_CW and CW_per_msg. The following table shows a number of different values these parameters can take to achieve different ECMG requirements.

Example	Requirements	lead_CW	CW_per_msg
1	1 CW per ECM per CP	0	1
2	the CWs for the current and next CP per ECM and the ECMG buffers the current CW from the previous CW Provision message	1	1
3	the CWs for the current and next CP per ECM and the ECMG receives both CWs from the SCS in each CW Provision message	1	2
4	3 CWs per ECM per CP	1	3

These graphs depict which CWs has to be passed for a specific CP, based on the different methods listed above. For any given CP, X-axis, the corresponding CW is portrayed on the Y-axis. In the CW_provision message, the boxed CWs are the set of CP_CW_combination that has to be passed.



Example 1



Example 2

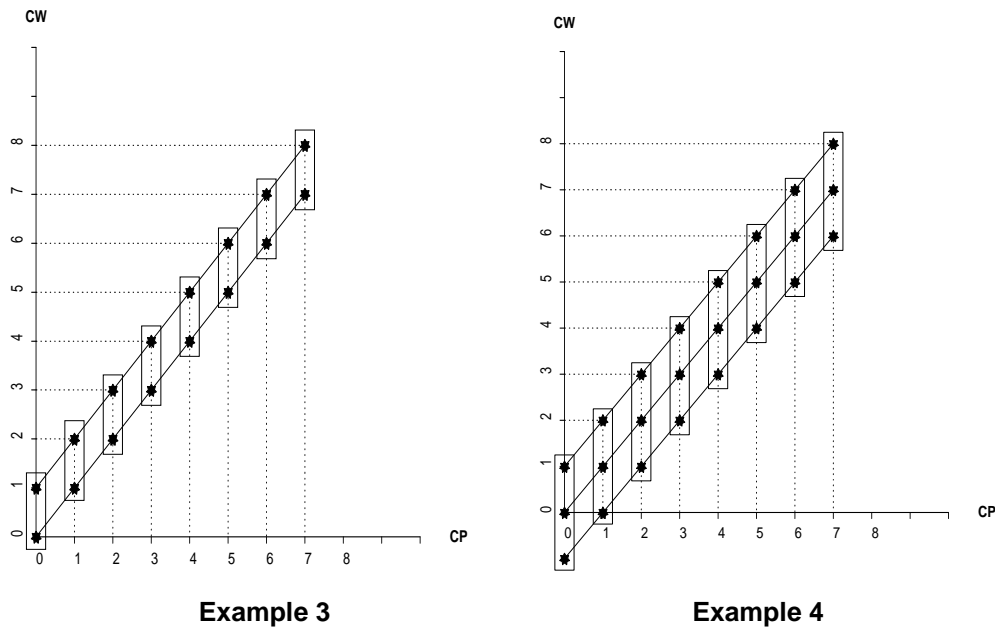


Figure 2: Lead_CW to CW_per_msg relationship

5.5.8 ECM_response message: ECMG \Rightarrow SCS

Parameter	Number of instances in message
ECM_channel_id	1
ECM_stream_id	1
CP_number	1
ECM_datagram	1

The ECM_response message (message_type 0x202) is a reply to the CW_provision message. It carries the ECM datagram, computed by the ECMG, from the information provided by the CW_provision message (and possibly from other CA specific information). The value of the CP_number parameter shall be the same in the replied ECM_response message as in the previous incoming CW_provision message (on that stream).

The time-out for the ECM_response message shall be computed by the SCS from the max_comp_time value defined during channel setup, and the typical network delays.

5.6 Error status

NOTE: TCP connection level errors are beyond the scope of the present document. Only channel, stream and application level errors are dealt with. These errors occur during the lifetime of a TCP connection.

There are two different error messages on these interfaces. The channel_error message for channel wide errors and the stream_error message for stream specific errors. These messages are sent by the ECMG to the SCS. When the ECMG reports an error to the SCS, it is up to the SCS to decide the most appropriate step to be taken. However "unrecoverable error" explicitly means that the channel or stream (depending on the message used) has to be closed. Most of the error status listed in the table 6 cannot occur in normal operation. They are mainly provided to facilitate the integration and debugging phase.

Table 6: ECMG protocol error values

error_status value	Error type
0x0000	DVB Reserved
0x0001	invalid message
0x0002	unsupported protocol version
0x0003	unknown message_type value
0x0004	message too long
0x0005	unknown Super_CAS_id value
0x0006	unknown ECM_channel_id value
0x0007	unknown ECM_stream_id value
0x0008	too many channels on this ECMG
0x0009	too many ECM streams on this channel
0x000A	too many ECM streams on this ECMG
0x000B	not enough control words to compute ECM
0x000C	ECMG out of storage capacity
0x000D	ECMG out of computational resources
0x000E	unknown parameter_type value
0x000F	inconsistent length for DVB parameter
0x0010	missing mandatory DVB parameter
0x0011	invalid value for DVB parameter
0x0012	unknown ECM_id value
0x0013	ECM_channel_id value already in use
0x0014	ECM_stream_id value already in use
0x0015	ECM_id value already in use
0x0016 to 0x6FFF	DVB Reserved
0x7000	unknown error
0x7001	unrecoverable error
0x7002 to 0x7FFF	DVB Reserved
0x8000 to 0xFFFF	ECMG specific/CA system specific/User defined

5.7 Security in ECMG \Leftrightarrow SCS protocol

The control words conveyed in the CP_CW_combination parameter within the CW_provision message constitute the clear cryptographic keys that are used to directly scramble content. Knowledge of these keys by unauthorized agents can result in the compromise of the security of the broadcast service. Therefore it is incumbent upon all Simulcrypt participants to employ effective and appropriate methods to preserve the confidentiality of the control words traversing this interface. One approach is to use only an inherently secure network for the ECMG \Leftrightarrow SCS interface. Another is to use a control word encryption scheme such as the one recommended in annex D of the present document to deliver the CW to the ECMG in a secure manner. In any case, the security of the CW on this interface shall be maintained so that unauthorized interception is prevented.

6 EMMG \Leftrightarrow MUX and PDG \Leftrightarrow MUX interfaces

6.1 Transport layer protocols for EMMG/PDG \Leftrightarrow MUX interfaces

To facilitate co-existence the DVB Simulcrypt Specification provides both TCP and UDP protocols for the EMMG/PDG \Leftrightarrow MUX interface. This co-existence is not required within the same head-end. In certain head-ends the UDP protocol may be more suitable (e.g. for network performance reasons) and in other head-ends the TCP protocol may be more suitable (e.g. for network reliability reasons). Therefore it is the head-end operator that decides which protocol is more appropriate and should be followed.

6.2 TCP-based protocol

6.2.1 Interface principles

6.2.1.1 Channel and Stream specific messages

The interface shall carry the following channel messages defined further in clause 6.2.4:

- *Channel_setup*
- *Channel_test*
- *Channel_status*
- *Channel_close*
- *Channel_error*

and the following stream messages defined further in clause 6.2.5:

- *Stream_setup*
- *Stream_test*
- *Stream_status*
- *Stream_close_request*
- *Stream_close_response*
- *Stream_error*
- *Data_provision*

For this interface, the EMMG/PDG is the client and the MUX is the server. In this TCP-based protocol, all messages are sent using TCP.

6.2.1.2 Channel establishment

The EMMG/PDG sends a *channel_setup* message to the MUX. In case of success the MUX replies by sending back a *channel_status* message.

In case of a rejection or a failure during channel setup the MUX replies with a *channel_error* message. This means that the channel has not been opened by the MUX and the EMMG/PDG shall close the TCP connection.

6.2.1.3 Stream establishment

The EMMG/PDG sends a *stream_setup* message to the MUX. In case of success the MUX replies by sending back a *stream_status* message. In case of a rejection or a failure the MUX replies with a *stream_error* message.

When a new EMM/private data stream is created in a transport stream, a new *data_id* shall be assigned to it by the CAS, according to the operational context (EMM/private data stream creation or EMMG/PDG replacement). The value of the *data_id* parameter remains unmodified as long as the EMM/private data stream exists. The combination {stream type + client_id + data_id} identifies uniquely this new EMM/private data stream in the whole system.

Once the connection, channel and stream have been correctly established the EMMs/private data will be transferred. They can be transferred as sections or as TS packets. The EMMG/PDG indicates at channel setup which kind of data object will be used. The EMMs/private data shall be inserted in the transport stream in the same order as they are provided by the EMMG/PDG.

6.2.1.4 Bandwidth allocation

The interface allows bandwidth negotiation between the EMMG/PDG and the MUX. This is not mandatory. During stream setup the EMMG/PDG will request the optimal bandwidth for that stream. The MUX will then respond with the bandwidth that has been allocated for that stream. The EMMG/PDG can, at a later time, request an adjustment in the bandwidth allocation. The MUX could also initiate an allocation change, without any request from the EMMG/PDG.

6.2.1.5 Stream closure

Stream closure is always initiated by the EMMG/PDG. This can occur when an EMM/private data stream is no longer needed. This is done by means of a `stream_close_request` message. A `stream_close_response` message indicates the stream has been closed.

6.2.1.6 Channel closure

Channel closure can occur when the channel is no longer needed or in case of error (detected by EMMG/PDG or reported by MUX). This is done by means of a `channel_close` message sent by the EMMG/PDG. Subsequently, the connection shall be closed by both sides.

6.2.1.7 Channel/Stream testing and status

At any moment either component can send a `channel_test/stream_test` message to check the integrity of a channel/stream. In response to this message the receiving component shall reply with either a channel/stream status message or a channel/stream error message.

6.2.1.8 Unexpected connection loss

Both EMMG/PDG and MUX shall be able to handle an unexpected communication loss (either on the connection, channel or stream level).

Each component, when suspecting a possible communication loss (e.g. a 10 second silent period), should check the communication status by sending a test message and expecting to receive a status message. If the status message is not received in a given time (implementation specific) the communication path should be re-established.

6.2.1.9 Handling data inconsistencies

If the MUX detects an inconsistency it shall send an error message to the EMMG/PDG. If the EMMG/PDG receives such a message or detects an inconsistency it should close the connection. The EMMG/PDG (as the client) will then (re-) establish the connection, channel and (if applicable) streams.

The occurrence of a user defined or unknown `parameter_type` or `message_type` shall not be considered as an inconsistency.

6.2.2 Parameter Type Values

Table 7: EMMG/PDG protocol parameter_type values

Parameter_type value	Parameter type	Type/Units	Length (bytes)
0x0000	DVB Reserved	-	-
0x0001	client_id	uimsbf	4
0x0002	section_TSpkt_flag	uimsbf	1
0x0003	data_channel_id	uimsbf	2
0x0004	data_stream_id	uimsbf	2
0x0005	datagram	user defined	variable
0x0006	bandwidth	uimsbf/kbit/s	2
0x0007	data_type	uimsbf	1
0x0008	data_id	uimsbf	2
0x0009 to 0x6FFF	DVB Reserved	-	-
0x7000	error_status	see clause 6.2.6	2
0x7001	error_information	user defined	variable
0x7002 to 0x7FFF	DVB reserved	-	-
0x8000 to 0xFFFF	user defined	-	-

6.2.3 Parameter semantics

bandwidth: this parameter is used in stream_BW_request and stream_BW_allocation messages to indicate the requested bit rate or the allocated bit rate respectively. It is the responsibility of the EMMG/PDG to maintain the bit rate generated within the limits specified by the MUX when the bandwidth allocation method is used (optional). It should be noted that the EMMG/PDG will operate from 0 kbit/s to the negotiated rate. The EMMG/PDG will not exceed the negotiated rate. If the bandwidth allocation method is not used the responsibility of bit rate control is not defined in the present document.

client_id: the client_id is a 32-bit identifier. It shall identify uniquely an EMMG/PDG across all the EMMGs/PDGs connected to a given MUX. To facilitate uniqueness of this value, the following rules apply:

- in the case of EMMs or other CA related data, the two first bytes of the client_id should be equal to the two bytes of the corresponding CA_system_id;
- in other cases a value allocated by DVB for this purpose should be used.

data_stream_id: this identifier uniquely identifies an EMM/private data stream within a channel.

data_channel_id: this identifier uniquely identifies an EMM/private data channel within a client_id.

data_id: the data_id is allocated by the CAS and uniquely identifies an EMM/private data stream of a client_id. The combination of the client_id and the data_id identifies uniquely an EMM/private data stream in the whole system. The unique identifier principle is described in clause 8.2.7.

data_type: type of data carried in the datagram in the stream:

- **0x00:** EMM;
- **0x01:** private data;
- **0x02:** DVB reserved (ECM);
- **other values:** DVB reserved.

datagram: this is the EMM/private data. According to the value of section_TSpkt_flag the Datagram can be transferred in either section format, TS packet format, or an arbitrary length datagram in accordance with the IP Datacast over DVB-H Service Purchase and Protection specification [26] (refer to annex N).

error_status: see clause 6.2.6.

error_information: this optional parameter contains user defined data completing the information provided by error_status. It can be an ASCII text or the parameter ID value of a faulty parameter for example.

section_TSpkt_flag: this parameter defines the format of the EMMs or of the private datagrams carried on this interface:

- **0x00:** the EMMs or private datagrams are in MPEG-2 section format;
- **0x01:** the EMMs or private datagrams are in MPEG-2 transport stream packet format; all TS packets shall be 188 byte long, any other payload length being considered as an error; it is the head-end's responsibility to fill the PID field in TS packet header;
- **0x02:** the EMMs carried on this interface are arbitrary length EMMs/KMMs in accordance with the IP Datacast Service Purchase and Protection specification [26] (refer to annex N);
- **other values:** DVB reserved.

6.2.4 Channel specific messages

6.2.4.1 Channel_setup message: EMMG/PDG ⇒ MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
section_TSpkt_flag	1

The channel_setup message (message_type = 0x0011) is sent by the EMMG/PDG to the MUX to setup a channel once the TCP connection has been established, as described in clause 6.2.1.2. It shall contain the client_id parameter indicating to the MUX the EMMG/PDG that is opening the channel.

6.2.4.2 Channel_test message: EMMG/PDG ⇔ MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1

The channel_test message (message_type = 0x0012) can be sent at any moment by either side to check:

- the channel is in an error free situation;
- the TCP connection is alive.

The peer shall reply with a channel_status message if the channel is free of errors, or a channel_error message if errors occurred.

6.2.4.3 Channel_status message: EMMG/PDG ⇔ MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
section_TSpkt_flag	1

The channel_status message (message_type = 0x0013) is a reply to the channel_setup message or the channel_test message (see clauses 6.2.1.2 and 6.2.1.7). All the parameters listed above are mandatory.

When the message is a response to a set-up, the values of the parameters are those requested by the EMMG/PDG and currently stored in the MUX. All these parameter values will be valid during the whole life time of the channel, for all the streams running on it.

When the message is a response to a test, the values of the parameters shall be those currently valid in the channel.

6.2.4.4 Channel_close message: EMMG/PDG \Rightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1

The channel_close message (message_type = 0x0014) is sent by the EMMG/PDG to indicate the channel is to be closed.

6.2.4.5 Channel_error message: EMMG/PDG \Leftrightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
error_status	1 to n
error_information	0 to n

A channel_error message (message type = 0x0015) is sent by the recipient of a channel_test message or by the MUX at any time to indicate that an unrecoverable channel level error occurred. A table of possible error conditions can be found in clause 6.2.6.

6.2.5 Stream specific messages

6.2.5.1 Stream_setup message: EMMG/PDG \Rightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1
data_id	1
data_type	1

The stream_setup message (message_type = 0x0111) is sent by the EMMG/PDG to setup a stream once the channel has been established, as described in clause 6.2.1.3.

6.2.5.2 Stream_test message: EMMG/PDG \Leftrightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1

The stream_test message (message_type = 0x0112) is used to request a stream_status message for the given client_id, data_channel_id and data_stream_id. The stream_test message can be sent at any moment by either entity. The peer shall reply with a stream_status message if the stream is free of errors, or a stream_error message if errors occurred.

6.2.5.3 Stream_status message: EMMG/PDG \Leftrightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1
data_id	1
data_type	1

The stream_status message (message_type = 0x0113) is a reply to the stream_setup message or the stream_test message.

The values of the parameters shall be those currently valid in the stream.

6.2.5.4 Stream_close_request message: EMMG/PDG \Rightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1

The stream_close_request message (message_type = 0x0114) is sent by the EMMG/PDG to indicate the stream is to be closed.

6.2.5.5 Stream_close_response message: EMMG/PDG \Leftarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1

The stream_close_response message (message_type = 0x0115) is sent by the MUX indicating the stream that is being closed.

6.2.5.6 Stream_error message: EMMG/PDG \Leftrightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1
error_status	1 to n
error_information	0 to n

A stream_error message (message type = 0x0116) is sent by the recipient of a stream_test message or by the MUX at any time to indicate that an unrecoverable stream level error occurred. A table of possible error conditions can be found in clause 6.2.6.

6.2.5.7 Stream_BW_request message: EMMG/PDG \Rightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1
bandwidth	0 to 1

The stream_BW_request message (message type = 0x0117) is always sent by the EMMG/PDG and can be used in two ways.

If the bandwidth parameter is present the message is a request for the indicated amount of bandwidth.

If the bandwidth parameter is not present the message is just a request for information about the currently allocated bandwidth. The MUX shall always reply to this message with a stream_BW_allocation message.

6.2.5.8 Stream_BW_allocation message: EMMG/PDG \Leftarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	1
data_stream_id	1
bandwidth	0 to 1

The stream_BW_allocation message (message type = 0x0118) is used to inform the EMMG/PDG about the bandwidth allocated. This can be a response to a stream_BW_request message or as a notification of a change in bandwidth initiated by the MUX. The message is always sent by the MUX.

If the bandwidth parameter is not present it means that the allocated bandwidth is not known.

NOTE: The bandwidth allocation message may indicate a different bandwidth than was requested (this could be less).

6.2.5.9 Data_provision message: EMMG/PDG \Rightarrow MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	0 to 1
data_stream_id	0 to 1
data_id	1
datagram	1 to n

The data_provision message is used by the EMMG/PDG to send, on a given data_stream_id, the datagram (in the case of EMMG this is EMM data).

In the TCP-based protocol, the data_provision message shall include the data_channel_id and data_stream_id parameters. In the UDP-based protocol (see clause 6.3), the data_provision message shall not include the data_channel_id and data_stream_id parameters.

6.2.6 Error status

NOTE: TCP connection level errors are beyond the scope of the present document. Only channel, stream and application level errors are dealt with. These errors occur during the lifetime of a TCP connection.

There are two different error messages on that interface. The channel_error message for channel wide errors, and the stream_error message for stream specific errors. These messages are sent by the MUX to the EMMG/PDG. When the MUX reports an error to the EMMG/PDG, it is up to the EMMG/PDG to decide the most appropriate step to be taken. However "unrecoverable error" explicitly means that the channel or stream (depending on the message used) has to be closed. Most error status listed in table 8 cannot occur in normal operation. They are mainly provided to facilitate the integration and debugging phase.

Table 8: EMMG/PDG protocol error values

error_status value	Error type
0x0000	DVB Reserved
0x0001	invalid message
0x0002	unsupported protocol version
0x0003	unknown message_type value
0x0004	message too long
0x0005	unknown data_stream_id value
0x0006	unknown data_channel_id value
0x0007	too many channels on this MUX
0x0008	too many data streams on this channel
0x0009	too many data streams on this MUX
0x000A	unknown parameter_type
0x000B	inconsistent length for DVB parameter
0x000C	missing mandatory DVB parameter
0x000D	invalid value for DVB parameter
0x000E	unknown client_id value
0x000F	exceeded bandwidth
0x0010	unknown data_id value
0x0011	data_channel_id value already in use
0x0012	data_stream_id value already in use
0x0013	data_id value already in use
0x0014	client_id value already in use
0x0015 to 0x6FFF	DVB Reserved
0x7000	unknown error
0x7001	unrecoverable error
0x7002 to 0x7FFF	DVB Reserved
0x8000 to 0xFFFF	MUX specific/CA system specific/User defined

6.3 UDP-based protocol

The EMMG/PDG UDP-based protocol is using the same message format as described in clause 6.2. Thanks to UDP functionality, it offers the unique advantage of broadcasting packets over a network and requires less network overhead; however, the EMMG/PDG UDP-based protocol does not provide additional feature to increase the intrinsic reliability of UDP.

6.3.1 Interface principles

In this protocol, only the data_provision message is sent using UDP/IP; it is the same message as the one used in the TCP-based protocol described in clause 6.2.5.9.

For the control part of this protocol, two methods can be used. The implementation of at least one of them is mandatory:

- the first method consists of using a TCP/IP connection supporting channel and stream management, as described in clause 6.2.1. This connection carries also test, status, error and bandwidth negotiation messages. When UDP/IP broadcast is used to send EMMs or Private Data, it is the responsibility of the EMMG/PDG to open a TCP/IP connection with each multiplexer that needs to receive the UDP packets produced by the EMMG/PDG;
- the second method consists of using the Simulcrypt Management Framework described in clause 7. With this method, the Network Management System (NMS) is responsible for ensuring that the head-end components are configured properly to receive and process the EMM/Private data produced by the EMMG/PDG.

Figure 3 illustrates the TCP-based version and the UDP-based version of the EMMG/PDG protocol.

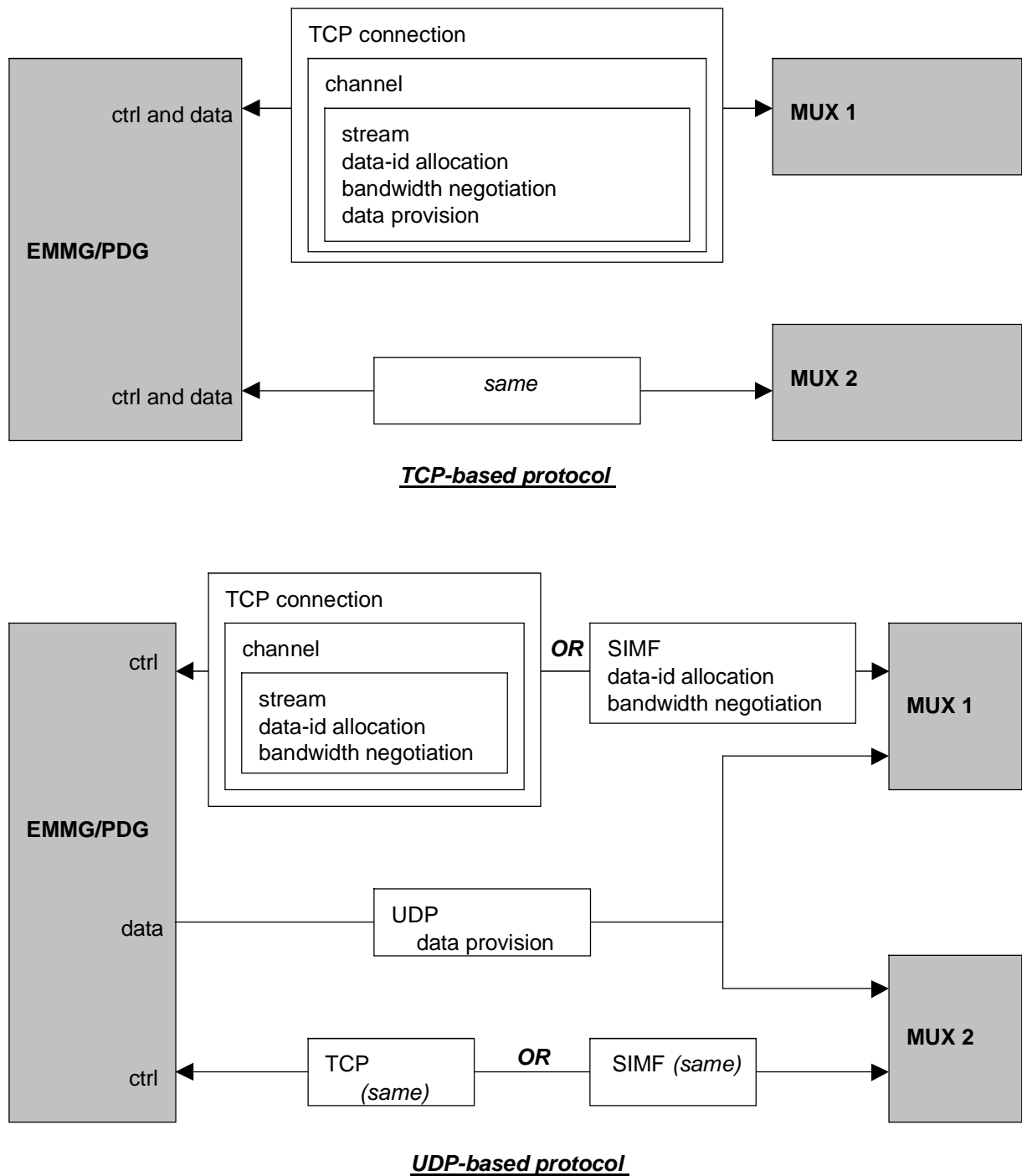


Figure 3: TCP-based and UDP-based EMMG/PDG protocols

6.3.1.1 Data_provision message: EMMG/PDG ⇒ MUX

Parameter	Number of instances in message
client_id	1
data_channel_id	0 to 1
data_stream_id	0 to 1
data-id	1
datagram	1 to n

The data provision message is used by the EMMG/PDG to send, on a given data_id the datagram (in the case of EMMG this is EMM data). Data_channel_id and data_stream_id are optional parameters. The client_id/data_id pair shall identify in a unique manner an EMM/private data stream across the system. For example, if two EMMGs send an EMM stream with the same data_id to the same multiplexer port, the multiplexer shall be able to distinguish between the two streams by looking at the client_id field in the data_provision message.

This message is the only message sent over UDP/IP and can be broadcast to several multiplexers.

6.3.1.2 Channel and stream configuration messages

Except for the data_provision message, all messages described in clauses 6.2.4 and 6.2.5 can be used on a separate TCP/IP connection to manage channels and streams. If a broadcast mechanism is used to send data_provision messages, the client_id, data_stream_id and data_id parameters will be the same for all the Multiplexers processing the EMM/Private data stream.

For this interface, the EMMG is the client, and the MUX is the server. Please refer to clause 6.2 (TCP-based protocol) for syntax details.

6.3.2 Bandwidth management

When using channel and stream configuration messages with UDP/IP broadcast data_provision messages, the same bandwidth negotiation messages need to be sent to all Multiplexers processing the EMM/private data stream:

- where TCP connections are used to manage configurations, it is the responsibility of the EMMG/PDG to ensure that all Multiplexers accept a new bandwidth configuration before changing the actual EMM/private data bandwidth. Multiplexers cannot be held responsible for overflow conditions on UDP/IP sockets;
- where SIMF is being used to manage configurations, it is the responsibility of the Network Management System (NMS) to ensure that all configuration changes are synchronized properly among the network components. For example, bandwidth increases shall first occur on the Multiplexers, while bandwidth decreases shall first occur on the EMMG/PDG to avoid loss of data.

7 Network management

7.1 SIMF overview

The Simulcrypt Integrated Management Framework (SIMF) addresses the requirements for interoperability between management components of multiple Conditional Access Systems (CASs) at a head-end. The framework does not address all of the issues relevant for a complete integrated Simulcrypt Management System. It specifies only the minimum set of components necessary to enable integration.

All Simulcrypt CAS/head-end interface specifications such as ECMG \leftrightarrow SCS, EMMG \leftrightarrow MUX, PDG \leftrightarrow MUX, and C(P)SIG \leftrightarrow (P)SIG are defined based on a connection-oriented communications paradigm (i.e. channel/streams). Alternatively, all these interfaces can be implemented using a transaction-based communications paradigm using the SIMF. In the current Simulcrypt specification this alternative is only available for the C(P)SIG \leftrightarrow (P)SIG interface.

The basic specification principle of the SIMF is:

- to define a common information model for management of all Simulcrypt conditional access components within the head-end, which are directly related to the Simulcrypt protocol. Based on the Simulcrypt Common Information Model (CIM) the Simulcrypt Management Information Base (SMIB) is defined. The CIM defines management information specific to Simulcrypt conditional access components of a head-end such as the EMM Generators (EMMGs), ECM Generators (ECMGs), Custom Service Information Generators (CSIG) and Custom Program Specific Information Generators (CPSIG). The CIM also defines generic information enabling management facilitating functions such as event and alarm specification and logging;
- to define a management protocol for manager/agent communication.

7.1.1 Introduction to the Common Information Model (CIM)

The CIM provides a sufficiently large common denominator to ease the integration of basic management functions fault, configuration, and performance management into a head-end network manager or a conditional access system manager. Specifically, the CIM consists of the following information:

- 1) configuration and status information of the following Simulcrypt conditional access components:
 - EMM Generators (EMMG);
 - Private Data Generators (PDG);
 - ECM Generators (ECMG);
 - (P)SI Generator ((P)SIG);
 - C(P)SI Generator (C(P)SIG).
- 2) generic event reporting information including:
 - event specification information;
 - alarm, state change, and value change notifications;
 - event forwarding information.
- 3) generic log control mechanism including:
 - log specification information;
 - alarm, state change, value change logs;
 - log filtering information.

The CIM is implemented in the Simulcrypt Management Information Base (SMIB), which within the framework is realized in an open-ended fashion. This is necessary to not constrain a particular head-end in the choice of implementation technologies but to also simultaneously enable integration by providing a standard MIB.

The management functions themselves are not mandated but are only enabled.

Thus, the DVB Simulcrypt Integrated Management Framework (SIMF) standardizes management information access via a management protocol but does not specify how the management information is to be used. The framework consists of the following:

- 1) The DVB Simulcrypt Management Information Base (SMIB) - using the basic concepts and vocabulary of the ITU-T TMN Information Model and the standard Internet SNMPv2 SMI, the SMIB consists of four modules:
 - MIB II as defined in RFC 1213 [2];
 - the Simulcrypt Identification Module (SIM);
 - the Simulcrypt Events Module (SEM);
 - the Simulcrypt Logs Module (SLM).
- 2) The management protocol to be used.

A CIM description is given in clauses 7.2 and 7.3.

7.1.2 SIMF specialization options

The framework enables the design and implementation of a complete integrated management system by specifying the common management information in the SMIB and a management protocol. To enable maximum flexibility in choosing the most appropriate technology for individual head-ends and facilitate open-endedness the following two specializations of the SIMF are defined of which one shall be chosen if a management system is to be supported by a head-end (it is the head-end operator who decides which options are more appropriate):

Option 1 - SNMP:

- the SMIB is realized as an SNMPv2 SMI;
- the management protocol is SNMPv2.

Option 2 - CORBA (not available for agents) (see note):

- the SMIB is realized as an IDL translation of the Option 1 SMI SMIB using the Joint Inter-Domain Management (JIDM) specification;
- the management protocol is defined as the SNMPv2 equivalent based on the JIDM specification.

NOTE: There is no obligation on a head-end management system to support CORBA agents. Only SNMP agents are always supported if the Simulcrypt Integrated Management Framework is implemented.

The two options facilitate a variety of possible DVB Simulcrypt Management Systems including all current major approaches as follows:

- 1) CORBA, TMN, TINA-C - see the OMG White Paper "CORBA-BASED Telecommunication Network Management Systems" which relies on the JIDM specification;
- 2) JMAPI - JMAPI supports both SNMP and CORBA;
- 3) WBEM - WBEM builds on top of existing frameworks.

The present document fully defines only the SNMPv2 and SNMPv2 SMI based SMIB.

The Simulcrypt Identification Module (SIM), the Simulcrypt Events Module (SEM) and the Simulcrypt Logs Module (SLM) can be used to support configuration, performance, and fault management of Simulcrypt components such as EMMG, ECMG, etc. While SIM is Simulcrypt specific, SEM and SLM also support generic notifications and logs and are applicable to any proprietary information modules. As such they can extend management functionality to the proprietary components if so desired by the component provider and the head-end facilitator.

7.2 Common Information Model (CIM)

The Common Information Model (CIM) definition is closely tied to SNMPv2 and SNMPv2 SMI, as this is the first option for a network management system realization, which can be transformed into the second one by means of the JIDM process. The Common Information Model specification maps directly into an SNMPv2 SMI MIB and therefore also includes the Internet Assigned Numbers Authority (IANA) Simulcrypt object number assignments.

The Common Information Model consists of four modules:

- MIB II as defined in RFC 1213 [2];
- the Simulcrypt Identification Module (SIM);
- the Simulcrypt Events Module (SEM);
- the Simulcrypt Logs Module (SLM).

The Simulcrypt Identification Module (SIM) reflects the Simulcrypt interfaces specified (e.g. ECMG/SCS). Therefore it is specified after these interfaces have been specified. The other three modules are used in all systems and are the pre-requisite for management of any Simulcrypt interface or the implementation of a Simulcrypt interface using the transaction based approach, i.e. C(P)SI/(P)SI interface.

The individual MIB II and SIM modules shall be implemented by Simulcrypt CA components if an integrated network management system is desired at the head-end and if a component is to be managed or monitored or if the interface is to be implemented using the transaction based transport. The individual SEM and SLM modules are optional and are the recommended solutions for event management and log management respectively. The technology underlying the module implementation can be different from component to component provided a JIDM based translator is provided to the common denominator platform of the head-end. For example, if the common denominator platform is CORBA and the module is implemented in an ECM generator as an SNMP MIB then a JIDM gateway needs to be provided between CORBA/IOP and SNMP. Not all network elements will have the need for all information groups defined in the modules.

7.2.1 Object Containment Hierarchy

Objects are unambiguously identified (or named) by assigning them an Object Identifier (OID). OIDs are globally unique for the entire Internet and are defined as a sequence of non-negative integers organized hierarchically similar to UNIX or DOS file system names. Each sequence element is also associated a textual name for ease of use. The last sequence name is commonly used by itself as a shorthand way of naming an object. Although there is no uniqueness requirement for shorthand names within the Internet-standard framework, by convention, an attempt is made to make those names unique by using a different prefix for objects in each new MIB. Figure 4 illustrates the Internet OID tree.

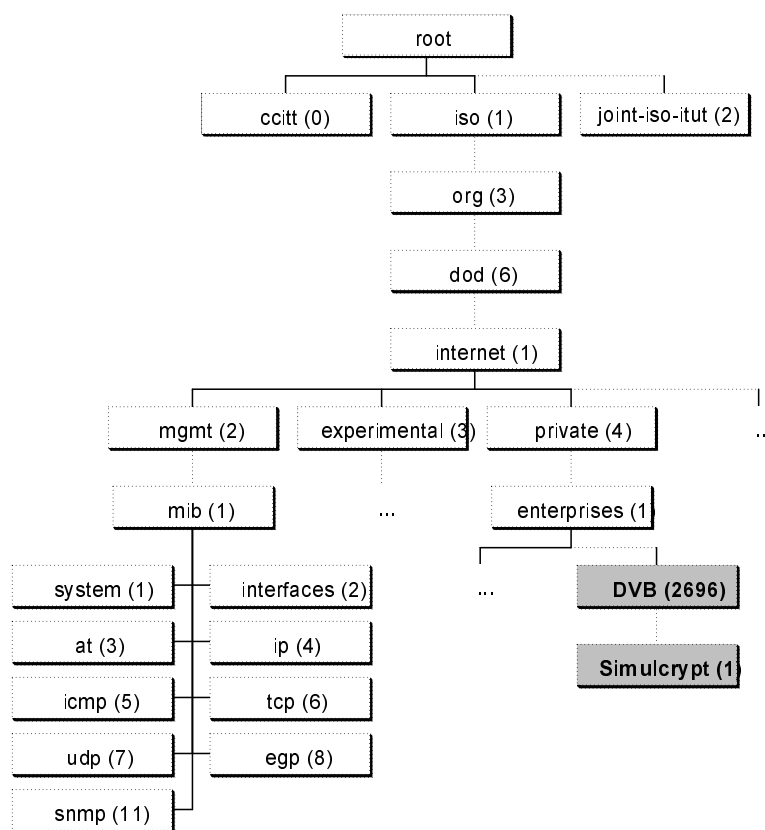


Figure 4: The Standard Internet OID Tree

Different OID formats can be used by interchanging the component names and numbers. For example:

- {iso org(3) dod(6) internet(1)} and 1.3.6.1 define the same object which is the Internet;
- {internet 4} and 1.3.6.1.4 define the same object which is the Private branch of the Internet subtree;
- {tcp 4} and 1.3.6.1.1.2.1.6.4 define the same object which is the TCP MIB module of the standard Internet management MIB.

For SNMP MIBs the following OID prefixes are important:

- internet defined as {iso(1) org(3) dod(6) 1};
- mgmt defined as {internet 2};
- experimental defined as {internet 3};
- private defined as {internet 4};
- mib, mib-1, and mib-2 which are defined as {mgmt 1};
- enterprises which is defined as {private 1}.

Objects for standard SNMP MIBs are defined under the "mib" branch of the hierarchy. "mib-1" has been superseded by "mib-2". Experimental MIBs being developed by IETF working groups define objects under the "experimental" branch. Proprietary MIBs define objects within an organization's subtree located under the "enterprises" branch (assigned by the Internet Assigned Numbers Authority - IANA).

The number assigned to DVB by IANA is **2696**. DVB Simulcrypt is assigned the first branch under DVB, which is **{enterprises 2696 1}** that corresponds to **1.3.6.1.4.1.2696.1**.

The DVB Simulcrypt MIB Modules are located as follows:

- {simMIB} which is 1.3.6.1.4.1.2696.1.1;
- {semMIB} which is 1.3.6.1.4.1.2696.1.2;
- {slmMIB} which is 1.3.6.1.4.1.2696.1.3.

Figure 5 illustrates the Simulcrypt MIB tree.

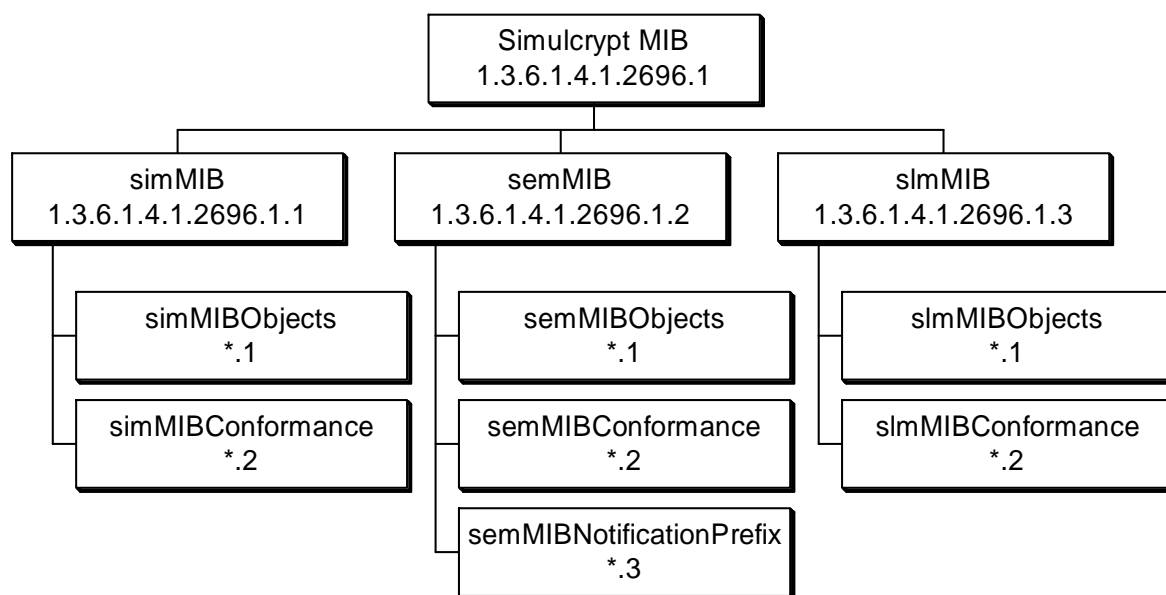


Figure 5: DVB Simulcrypt MIB Tree

7.2.2 MIB II

MIB II is a standard Internet MIB (see RFC 1213 [2]). Every SNMP agent is assumed to support MIB II. MIB-II, like its predecessor, the Internet-standard MIB, contains only essential elements. There is no need to allow individual objects to be optional. Rather, the objects are arranged into groups which are implemented by a system depending on whether the semantics of the group is applicable to an implementation. For example, an implementation has to implement the EGP group if and only if it implements the EGP. Following Groups are defined:

- **System:** Implementation of the System group is mandatory for all systems. The System group specifies things like system up time, system contact, system location, etc. If an agent is not configured to have a value for any of these variables, a string of length 0 is returned.
- **Interfaces:** Implementation of the Interfaces group is mandatory for all systems. This group defines the different sub network interfaces from this entity, i.e. Ethernet, FDDI, etc.
- **Address Translation (obsolete).**
- **IP:** Implementation of the IP Group is mandatory for all systems. This group specifies information related to IP like routing tables, address mappings, statistics, etc.
- **ICMP:** Implementation of the ICMP group is mandatory for all systems. This group specifies a number of different type of statistics related to ICMP.
- **TCP:** Implementation of the TCP group is mandatory for all systems that implement the TCP. This group contains TCP configuration information and information about the existing TCP connections. Note that instances of object types that represent information about a particular TCP connection are transient; they persist only as long as the connection in question.
- **UDP:** Implementation of the UDP group is mandatory for all systems, which implement the UDP. This group contains UDP related statistics as well as information of all current UDP listeners.
- **EGP:** Implementation of the EGP group is mandatory for all systems, which implement the EGP.
- **Transmission:** Based on the transmission media underlying each on a system, the corresponding portion of the Transmission group is mandatory for that system. When Internet-standard definitions for managing transmission media are defined, the transmission group used to provide a prefix for the names of those. Typically, such definitions reside in the portion of the MIB until they are "proven", then as a part of the Internet standardization process, definitions are accordingly elevated and a new identifier, under the transmission group is defined. By convention, the name assigned is: type OBJECT IDENTIFIER: = {transmission number}, where "type" is the symbolic value used for the media in the ifType column of the ifTable object, and "number" is the actual integer value corresponding to the symbol.
- **SNMP:** Implementation of the SNMP group is mandatory for all systems which support an SNMP protocol entity. Some of the objects defined will be zero-valued in those SNMP implementations that are optimized to support only those functions specific to either a management agent or a management station.

7.2.3 Concurrency control

Concurrency control of simultaneous updates of an agent's MIB variables by multiple managers is accomplished by using the SMIB SIM module's administrative state variables (as defined in the ITU-T Recommendation X.731 [4] standard used in TMN) and the SNMP administrative framework. An administrative state of a group of objects is either locked, unlocked, or shutting down. These states and the corresponding state transitions are defined in ITU-T Recommendation X.731 [4].

A manager can only lock a variable group if it is part of the agent's, module's, etc. management community and if the agent, module, etc. is not already locked. Once an agent is locked it is protected from access by other managers as any manager wanting to access the concurrency-controlled agent, module, etc. will first attempt to lock it and will fail if that is not possible (i.e. managers are trusted). If a manager should crash leaving the agent, module, etc. locked, the designated back-up managers within the same community can unlock the agent.

7.2.4 Simulcrypt Events Module (SEM)

SEM enables managers to configure the types of events that can be generated by an agent and when those events should be transformed into asynchronous notifications (SNMP Traps) to be sent to different managers. The mechanism and information model used are based on the ITU-T Recommendations X.733 [5] and X.734 [6] standards defining event management and alarm reporting. SEM defines the following object groups:

- **Events Group:** This group consists of event configuration information defining the types of events that the agent shall generate.
- **Event Forwarding Discriminator (EFD) Group:** This group consists of EFD configuration information defining what types of events an EFD will transform into notifications, at what times of day it will do so, and to which managers it will send the notifications to.
- **Event Notifications:** This group defines three types of notifications, which an agent can send to a manager. These are an alarm, a state change notification, and a value change notification. Each EFD specifies what type of notification is to be sent for an event that has occurred in the agent. The EFD also specifies the conditions under which such a notification is to be sent and the IP address of the manager to which the notification is to be sent. All standard SNMP traps are sent to the managers UDP port 162. Most management platforms support this mechanism.

The Events Group stores event descriptions in a table. Each row in the table corresponds to an event that the agent is to generate. The event description in that table specifies when the agent is to generate the event, i.e. because a MIB variable has crossed a specified threshold, because a state has been changed, etc.

Once the agent generates an event as specified in the Event table it checks the EFD table to find an EFD that matches that event and specifies what kind of notification is to be generated and to which manager that notification is to be sent. The match is performed based on event characteristics such as event type, etc.

The EFDs in the EFD table are controlled by three state/status variables, the administrative state, the operational state, and the availability status. If the administrative state is not unlocked or the operational state is not enabled or the availability status is not available, the EFD is inactive (that means it is ignored by the agent). The manager sets the administrative and operational states. The availability status is set as a result of an automatic scheduling function that is also associated with the EFD and specified in the EFD table. This scheduling function includes specifications of a daily start and stop time and a weekly mask specifying when the EFD changes availability status from off-duty to available.

Figure 6 illustrates the MIB tree of the Simulcrypt Events Module (SEM).

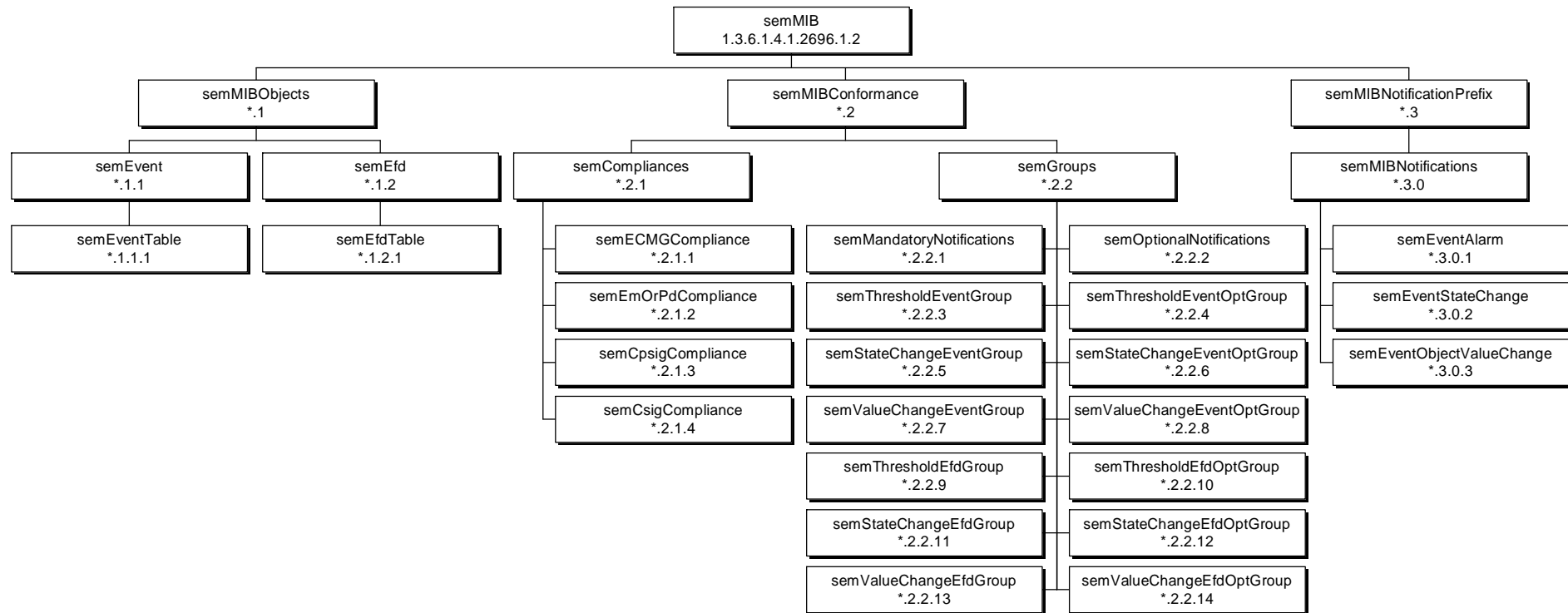


Figure 6: Simulcrypt Events Module (SEM)

7.2.4.1 Event Group

This group consists of event configuration information defining the types of events that the agent shall generate.

Table 9: CIM - SEM (P)SIG Group - Event Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
semEventName	bslbf/the unique name of the target event, EntryName (SNMP)	provides a unique identification of an event	read-create
semEventAdmin State	enumerated/administrative state of a table row, enumerated type (ITU-T Recommendation X.731 [4])	enables concurrency control between multiple management entities	read-create
semEventAlarm Status	enumerated/alarm status of an event, enumerated type (ITU-T Recommendation X.731 [4])	enables event monitoring and clearing	read-create
semEventType	enumerated/indicates the type of event, enumerated type (ITU-T Recommendation X.734 [6])	enables differentiation between distinct event types	read-create
semEventText	bslbf/a description of an event's function and use, ASCII string of maximum 256 characters	enables textual description of an event for human readers	read-create
semEventChanged Objectid	bslbf/the object identifier of the MIB object to check and see if the event should fire, OBJECT IDENTIFIER (SNMP)	enables association of MIB objects with events	read-create
semEventToState Change	4 uimbsf/if semEventChangedObjectid is a state/status variable this variable identifies the state that causes the event to be generated	enables association of events with state/status variables	read-create
semEventRising Threshold	4 uimbsf/if semEventChangedObjectid is a threshold variable this variable indicates the threshold value to check against; if the value of semEventChangedObjectid is greater than or equal an event is generated; 32-bit unsigned integer	enables association of events with threshold variables	read-create
semEventFalling Threshold	4 uimbsf/if semEventChangedObjectid is a threshold variable this variable indicates the threshold value to check against; if the value of semEventChangedObjectid is less than or equal an event is generated	enables association of events with threshold variables	read-create
semEventProbable Cause	enumerated/defines further probable cause for the last event of this type, enumerated type (ITU-T Recommendation X.734 [6])	enables differentiation between event causes	read
semEvent PerceivedSeverity	enumerated/defines the perceived severity of the last event of this type, enumerated type (ITU-T Recommendation X.734 [6])	enables differentiation of event severity level	read
SemEventTrend Indication	enumerated/indicates the trend of the last event of this type, enumerated type (ITU-T Recommendation X.734 [6])	enables the indication of the event trend, i.e. more/less severe	read
semEventBacked UpStatus	enumerated/indicates the backed up status of the object causing the event, enumerated type (ITU-T Recommendation X.731 [4])	enables identification of backed up objects	read-create
semEventBacked UpObject	bslbf/if the backed up status is backedUp this variable contains the object identifier of the back up object, OBJECT IDENTIFIER (SNMP)	enables specification of back up objects	read-create
semEventSpecific Problems	bslbf/identifies the object responsible for the problem, OBJECT IDENTIFIER (SNMP)	enables specification of specific problems	read-create
semEvent Frequency	4 uimbsf/identifies the number of seconds to wait between event frequency checks, 32-bit unsigned integer	enables event throttling	read-create
semEvent Sensitivity	enumerated/identifies whether the event is level or edge sensitive, Enumerated	enables two different types of event generation mechanisms: whenever a threshold is crossed and periodically as long as a threshold has been crossed	read-create
semEventStatus	enumerated/status variable for synchronizing row creation/deletion between management entities, RowStatus (SNMP)	enables synchronization of row creation/deletion	read-create

7.2.4.2 Event Forwarding Discriminator (EFD) Group

This group consists of EFD configuration information defining what types of events an EFD will transform into notifications, at what times of day it will do so, and to which managers it will send the notifications to. An EFD generates a notification if it is unlocked (administrative state), enabled (operational state) and available (availability status) and if all of the specified discriminators are true, i.e. if semEfdDiscriminatedTypes is specified then the type indicated has to match the type of the event for a notification to be generated. If multiple discriminators are specified by a single EFD then all have to be matched (i.e. logical AND) before a notification is generated. A not specified discriminator in an EFD is always TRUE. A single event can match multiple EFDs and generate multiple notifications if so specified by the semEfdOr variable.

Table 10 is indexed by the event name (semEfdName) and the target address of the management entity, which is to receive the notification (semEfdTarget).

Table 10: CIM - SEM (P)SIG Group - EFD Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
semEfdName	bslbf/the unique name of the EFD, EntryName (SNMP)	provides a unique identification of an EFD	read-create
semEfdAdminState	enumerated/administrative state of a table row, enumerated type (ITU-T Recommendation X.731 [4])	enables concurrency control between multiple management entities	read-create
semEfdOperState	enumerated/operational state of an EFD, enumerated type (ITU-T Recommendation X.731 [4])	enables the indication of the current operation state	read-create
semEfdAvailStatus	enumerated/reflects the scheduling of the EFD, enumerated type (ITU-T Recommendation X.731 [4])	enables scheduling	read
semEfdStartTime	bslbf/defines the date and time at which an unlocked and enabled EFD starts functioning, i.e. changes the availability status from offDuty to available, DateAndTime (SNMP)	enables the scheduling of EFDs	read-create
semEfdStopTime	bslbf/defines the date and time at which an unlocked and enabled EFD stops functioning, i.e. changes its availability status from available to offDuty, DateAndTime (SNMP)	same	read-create
semEfdDailyStartTime	bslbf/defines the daily start time at which an unlocked and enabled EFD starts functioning, i.e. changes its availability status from offDuty to available, TimeTicks (SNMP)	enables daily scheduling of EFDs	read-create
semEfdDailyStopTime	bslbf/defines the daily stop time at which an unlocked and enabled EFD stops functioning, i.e. changes its availability status from available to offDuty, TimeTicks (SNMP)	same	read-create
semEfdWeeklyMask	1 uimbsbf/defines the weekly schedule at which an unlocked and enabled EFD starts functioning, an octet string of 1 octet	enables weekly scheduling	read-create
semEfdTypes	enumerated/the event type that this EFD may generate notifications for, enumerated type (ITU-T Recommendation X.734 [6])	enables an EFD to be specialized for a particular event type	read-create
semEfdCause	enumerated/the probable cause that this EFD may generate notifications for, enumerated type (ITU-T Recommendation X.734 [6])	enables an EFD to be specialized by probable cause	read-create
semEfdSeverity	enumerated/the perceived severity that this EFD may generate notifications for, enumerated type (ITU-T Recommendation X.734 [6])	enables an EFD to be specialized by severity	read-create
semEfdSpecificProblems	bslbf/the identifier of the object that may cause the generation of a notification by this EFD, OBJECT IDENTIFIER (SNMP)	enables an EFD to be specialized by event causing Object	read-create
semEfdTrendIndication	enumerated/identifies the event trend that will cause a notification to be generated, enumerated type (ITU-T Recommendation X.734 [6])	enables an EFD to be specialized by event trend	read-create

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
semEfdChanged Objectld	bslbf/identifies the object whose value change shall cause the generation of a notification, OBJECT IDENTIFIER (SNMP)	enables an EFD to be specialized by value change of an object	read-create
semEfdToState Change	4 uimsbf/the to state of the object that may cause the generation of a notification by this EFD	enables an EFD to be specialized by a state value	read-create
semEfdNotification	bslbf/identifies the notification object identifier to be generated if conditions are met, OBJECT IDENTIFIER (SNMP)	enables the association of a notification type with an EFD	read-create
semEfdOr	enumerated/identifies whether the EFD table shall be searched further for other possible matches and further possible notification generation, enumerated	enables multiple notifications to be generated by an event	read-create
semEfdTarget	bslbf/the IP address of the management entity to receive the notification if generated, IpAddress (SNMP)	enables the specification of the target management entity for the notifications generated by the EFD	read-create
semEfdText	bslbf/a description of an event's function and use, ASCII string of maximum 256 characters	enables textual description of an EFD for human readers	read-create
semEfdStatus	enumerated/status variable for synchronizing row creation/deletion between management entities, RowStatus (SNMP)	enables synchronization of row creation/deletion	read-create

7.2.4.3 Event Notification Group

This group defines three types of notifications which an agent can send to a manager. These are an alarm, a state change notification, and a value change notification. Each EFD specifies what type of notification is to be sent for an event that has occurred in the agent. The EFD also specifies the conditions under which such a notification is to be sent and the IP address of the manager to which the notification is to be sent. All standard SNMP traps are sent to the managers UDP port 162. Most management platforms support this mechanism.

The first notification type that can be generated is a semEventAlarm, which carries in addition to the standard SNMPv2 notification parameters the following objects from the events table:

- semEventName;
- semEventType;
- semEventProbableCause;
- semEventSpecificProblems;
- semEventPerceivedSeverity;
- semEventTrendIndication;
- semEventText.

The second notification type that can be generated is a semEventStateChange, which carries in addition to the standard SNMPv2 notification parameters the following objects from the events table:

- semEventName;
- semEventStateChange;
- semEventChangedObjectId.

The third notification type that can be generated is a semEventObjectValueChange, which carries in addition to the standard SNMPv2 notification parameters the following objects from the events table:

- semEventName;
- semEventChangedObjectId.

7.2.4.4 Conformance requirements

The Simulcrypt Events Module (SEM) is optional. However if this module is implemented, table 9 summarizes the conformance requirements for management entities implementing the Simulcrypt Events Module (SEM).

Table 11: CIM - SEM Conformance Requirements

Common Information Model- Simulcrypt Events Module Group	Management Entity Hosting or Representing an ECMG		Management Entity Hosting or Representing an EMMG		Management Entity Hosting or Representing a PDG		Management Entity Hosting or Representing a CPSIG		Management Entity Hosting or Representing a CSIG	
	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional
Events Group - threshold events	name, type, text, object-id, thresholds, cause, severity, trend, frequency, status, admin state	backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, thresholds, cause, severity, trend, frequency, status, admin state	backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, thresholds, cause, severity, trend, frequency, status, admin state	backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, thresholds, cause, severity, trend, frequency, status, admin state	backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, thresholds, cause, severity, trend, frequency, status, admin state	backed up, back up-id, specific problems, sensitivity, alarm status
Events Group - state change events	name, type, text, object-id, to state, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, to state, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, to state, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, to state, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, to state, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status
Events Group - value change events	name, type, text, object-id, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status	name, type, text, object-id, frequency, status, admin state	cause, severity, trend, backed up, back up-id, specific problems, sensitivity, alarm status

Common Information Model-Simulcrypt Events Module Group	Management Entity Hosting or Representing an ECMG		Management Entity Hosting or Representing an EMMG		Management Entity Hosting or Representing a PDG		Management Entity Hosting or Representing a CPSIG		Management Entity Hosting or Representing a CSIG	
	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional
EFD Group - threshold events	name, admin state, oper state, avail status, types, cause, severity, trend, object-id, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, cause, severity, trend, object-id, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, cause, severity, trend, object-id, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, cause, severity, trend, object-id, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, cause, severity, trend, object-id, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type
EFD Group - state change events	name, admin state, oper state, avail status, types, object-id, to state, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, to state, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, to state, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, to state, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, to state, notification, target, status	start, stop, dstart, dstop, week, specific problems, text, notification type
Events Group - value change events	name, admin state, oper state, avail status, types, object-id, notification, target, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, notification, target, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, notification, target, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, notification, target, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems, text, notification type	name, admin state, oper state, avail status, types, object-id, notification, target, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems, text, notification type
Notifications Group	alarm	state change, value change	alarm	state change, value change	alarm	state change, Value change	alarm	state change, Value change	alarm	state change, Value change

7.2.5 Simulcrypt Logs Module (SLM)

SLM enables managers to configure the types of logs that can be generated by an agent. The mechanism and information model used are based on the ITU-T Recommendation X.735 [7] Log Model.

The log in addition to conceptually storing the logged information determines which information is to be logged. Each log contains a discriminator construct, which specifies the characteristics an event shall have in order to be selected for logging. SLM consists of the following object groups:

- Log Control Group: This group defines the types of logs tables the agent is maintaining, their discriminators, the log scheduling, etc.;
- The Logs Group: This group defines three logs: the alarm logs, the state change logs, and the object value change logs.

Logs are controlled by the Log Control Table as specified in the ITU-T Recommendation X.735 [7]. Each entry in that table associates events with logs and specifies when the event is to be logged in that log. The event is logged if the log discriminator holds. That is if the event is of a certain type, if it has been generated by a certain object, if it exceeds a certain threshold, etc. The log control entries themselves are controlled by state/status variables, the administrative state, the operational state, and the availability status. The manager can set the administrative and operational states. The availability status is set by the agent itself based on an automatic log control scheduling mechanism, which specifies the times during which the logs are to be made.

Log Control Table entries also specify log control information and log statistics.

The three logs defined are defined as tables in which each event is stored as a row. The logs in the alarm log table are logs of alarm events that have passed the log control discriminator in the Log Control Table. Similarly the logs in the state change log table are logs of state changes. And logs in the object value change table are logs of object value changes.

Figure 7 illustrates the Simulcrypt Logs Module MIB tree.

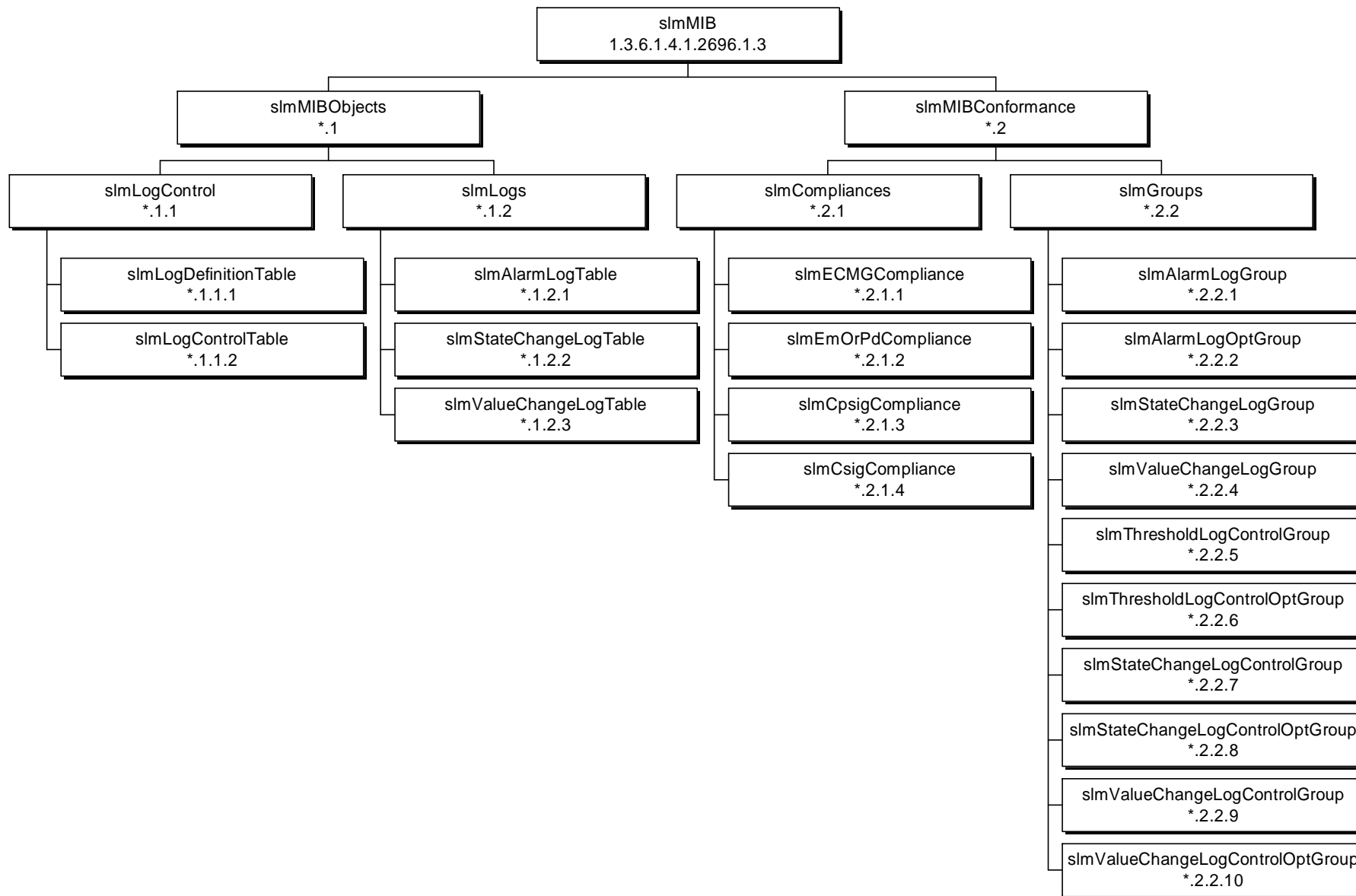


Figure 7: Simulcrypt Logs Module SLM

7.2.5.1 Log Control Group

This group defines the types of logs tables the agent is maintaining, their discriminators, the log scheduling, etc. The group consists of two tables, the Log Definition Table and the Log Control Table.

The Log Definition Table (table 12) is used to define all logs in the system. Each entry consists of an `slmLogDefinitionId`, which is the Log identifier, variables defining the log state, and a specification of the log full action. Table 12 is indexed by the `logDefinitionName`.

Table 12: CIM - SLM Log Definition Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
<code>slmLogDefinitionName</code>	bslbf/the unique name of the Log Definition Entry, <code>EntryName</code> (SNMP)	provides a unique identification of a Log Definition Entry	read-create
<code>slmLogDefinitionId</code>	bslbf/the unique log table identifier, <code>OBJECT IDENTIFIER</code> (SNMP)	provides a unique identification of Log Tables	read-create
<code>slmLogDefinitionAdminState</code>	enumerated/administrative state of a table row, enumerated (ITU-T Recommendation X.731 [4])	enables concurrency control between multiple management entities	read-create
<code>slmLogDefinitionOperState</code>	enumerated/operational state of an EFD, enumerated (ITU-T Recommendation X.731 [4])	enables the indication of the current operation state	read-create
<code>slmLogDefinitionAvailStatus</code>	enumerated/reflects the scheduling of the Log Control entry, enumerated (ITU-T Recommendation X.731 [4])	enables scheduling	read
<code>slmLogDefinitionFullAction</code>	enumerated/defines what action to take when the maximum log table size has been reached, enumerated (ITU-T Recommendation X.735 [7])	enables control of log full action	read-create
<code>slmLogDefinitionMaxLogSize</code>	4 uimsbf/defines the maximum size of a log table in number of octets	enables control of the maximum log table size	read-create
<code>slmLogDefinitionCurrentLogSize</code>	4 uimsbf/defines the current log table size	enables monitoring of log table size	read
<code>slmLogDefinitionNumberOfRecords</code>	4 uimsbf/specifies the number of log records in the log table	enables monitoring of the log table size	read

The Log Control Table (table 13) defines possibly multiple Log Controls for the different Logs. Each Log is identified by the Log Definition name. Each Log Control is identified by a Log Control name. Table 13 is indexed by the `LogDefinitionName` and the `LogControlName`. Each log control consists of scheduling and log filtering information.

Table 13: CIM - SLM Log Control Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
<code>slmLogDefinitionName</code>	bslbf/the unique name of the Log Definition Entry, <code>EntryName</code> (SNMP)	provides a unique identification of a Log Definition Entry	read-create
<code>slmLogControlName</code>	bslbf/the unique name of the Log Control Entry, <code>EntryName</code> (SNMP)	provides a unique identification of a Log Control Entry	read-create
<code>slmLogControlStartTime</code>	bslbf/defines the date and time at which an unlocked and enabled Log Control entry starts functioning, i.e. changes the availability status from <code>offDuty</code> to <code>available</code> , <code>DateAndTime</code> (SNMP)	enables the scheduling of Log Controls	read-create

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
slmLogControlStopTime	bslbf/defines the date and time at which an unlocked and enabled Log Control entry stops functioning, i.e. changes its availability status from available to offDuty, DateAndTime (SNMP)	same	read-create
slmLogControlDailyStartTime	bslbf/defines the daily start time at which an unlocked and enabled Log Control entry starts functioning, i.e. changes its availability status from offDuty to available, TimeTicks (SNMP)	enables daily scheduling of log control entries	read-create
slmLogControlDailyStopTime	bslbf/defines the daily stop time at which an unlocked and enabled Log Control entry stops functioning, i.e. changes its availability status from available to offDuty, TimeTicks (SNMP)	same	read-create
slmLogControlWeeklyMask	bslbf/defines the weekly schedule at which an unlocked and enabled Log Control entry starts functioning, octet string of length 1	enables weekly scheduling	read-create
slmLogControlTypes	enumerated/the event type that this Log Control entry may generate logs for, enumerated (ITU-T Recommendation X.734 [6])	enables a Log Control entry to be specialized for a particular event type	read-create
slmLogControlCause	enumerated/the probable cause that this Log Control entry may generate logs for, enumerated (ITU-T Recommendation X.734 [6])	enables an Log Control entry to be specialized by probable cause	read-create
slmLogControlSeverity	enumerated/the perceived severity that this Log Control entry may generate logs for, enumerated (ITU-T Recommendation X.734 [6])	enables an Log Control entry to be specialized by severity	read-create
slmLogControlSpecificProblems	bslbf/the identifier of the object that may cause the generation of a log entry by this Log Control entry, OBJECT IDENTIFIER (SNMP)	enables an Log Control entry to be specialized by Object	read-create
slmLogControlToStateChange	bslbf/the to state of the object that may cause the generation of a log entry by this Log Control entry, 32-bit unsigned integer	enables an Log Control entry to be specialized by a state value	read-create
slmLogControlTrendIndication	enumerated/identifies the trend that will cause a log entry to be made, enumerated (ITU-T Recommendation X.734 [6])	enables specialization of log control based on event trends	read-create
slmLogControlChangedObjectId	bslbf/identifies the object that changed value and should be logged, OBJECT IDENTIFIER (SNMP)	enables specialization of log control based on objects causing the event	read-create
slmLogControlStatus	enumerated/status variable for synchronizing row creation/deletion between management entities, RowStatus (SNMP)	enables synchronization of row creation/deletion	read-create

7.2.5.2 Logs Group

The three logs defined are defined as tables in which each event is stored as a row. The logs in the alarm log table are logs of alarm events that have passed the log control discriminator in the Log Control Table. Similarly the logs in the state change log table are logs of state changes. Logs in the object value change table are logs of object value changes. Table 14 defines logs of alarm events as follows.

Table 14: CIM - SLM Alarm Log Table

Object	Size/Description	Object Justification	Head-end/CA Maximum Access Right
slmAlarmLogName	bslbf/the unique name of the Log Control Entry, EntryName (SNMP)	provides a unique identification of the alarm log entry; it is identical to the event name	read
slmAlarmLogTime	4 uimbsbf/the time at which the alarm has been logged, TimeTicks (SNMP)	provides a unique identification of Log Tables	read
slmAlarmLogText	bslbf/a textual description of the event being logged, ASCII string of maximum 256 characters	records the event description	read
slmAlarmLogType	enumerated/the event type of this log entry, enumerated (ITU-T Recommendation X.734 [6])	records alarm type	read
slmAlarmLogCause	enumerated/the event cause of this log entry, enumerated (ITU-T Recommendation X.734 [6])	records event cause	read
slmAlarmLogSeverity	enumerated/the alarm severity of the logged event, enumerated (ITU-T Recommendation X.734 [6])	records event severity	read
slmAlarmLogSpecificProblems	bslbf/the identifier of the object that caused the logged event, OBJECT IDENTIFIER (SNMP)	records the id of objects causing the event	read
slmAlarmLogTrendIndication	enumerated/the trend of the event that has been logged, enumerated (ITU-T Recommendation X.734 [6])	records event trend	read
slmAlarmLogChangedObjectId	bslbf/identifies the object that changed value and caused the logged event, OBJECT IDENTIFIER (SNMP)	records the id of the object causing the event	read

Table 15 defines logs of state change events as follows.

Table 15: CIM - SLM State Change Log Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
slmStateChangeLogName	bslbf/the unique name of the Log Control Entry, EntryName (SNMP)	provides a unique identification of the log entry; it is identical to the event name	read
slmStateChangeLogTime	4 uimbsbf/the time at which the alarm has been logged, TimeTicks (SNMP)	provides a unique identification of Log Tables	read
slmStateChangeLogText	bslbf/a textual description of the event being logged, ASCII string of maximum 256 characters	records the event description	read
slmStateChangeLogToStateChange	bslbf/the to state change of the event being logged, enumerated (ITU-T Recommendation X.734 [6])	records event to state change	read
slmStateChangeLogChangedObjectId	bslbf/identifies the object that changed value and caused the logged event, OBJECT IDENTIFIER (SNMP)	records the id of the object causing the event	read

Table 16 defines logs of value change events as follows.

Table 16: CIM - SLM Value Change Log Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
slmValueChangeLogName	bslbf/the unique name of the Log Control Entry, EntryName (SNMP)	provides a unique identification of the log entry; it is identical to the event name	read
slmValueChangeLogTime	4 uimsbf/the time at which the alarm has been logged, TimeTicks (SNMP)	provides a unique identification of Log Tables	read
slmValueChangeLogText	bslbf/a textual description of the event being logged, ASCII string of maximum 256 characters	records the event description	read
slmValueChangeLogChangedObjectId	bslbf/identifies the object that changed value and caused the logged event, OBJECT IDENTIFIER (SNMP)	records the id of the object causing the event	read

7.2.5.3 Conformance Requirements

The Simulcrypt Logs Module (SLM) is optional. However if log management is implemented the following conformance requirements hold.

Table 17: CIM - SLM Conformance Requirements

Common Information Model - CIM Simulcrypt Logs Module Group	Management Entity Hosting or Representing an ECMG		Management Entity Hosting or Representing an EMMG/		Management Entity Hosting or Representing a PDG		Management Entity Hosting or Representing a CPSIG		Management Entity Hosting or Representing a CSIG	
	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional	mandatory	optional
Log Control Group - threshold events	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, cause, severity, trend, object-id, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, cause, severity, trend, object-id, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, cause, severity, trend, object-id, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, cause, severity, trend, object-id, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, cause, severity, trend, object-id, status	start, stop, dstart, dstop, week, specific problems
Log Control Group - state change events	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, to state, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, to state, status	start, stop, dstart, dstop, week, specific problems	admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, to state, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, to state, status	start, stop, dstart, dstop, week, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, to state, status	start, stop, dstart, dstop, week, specific problems
Log Control Group - value change events	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, status	name, log-id, start, stop, dstart, dstop, week, cause, severity, trend, specific problems	admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems	name, log-id, admin state, oper state, avail status, full action, max log, number of recs, current log size, types, object-id, status	start, stop, dstart, dstop, week, cause, severity, trend, specific problems
Logs Group - alarm logs	name, time, text, type, object-id	severity, problems, trend	name, time, text, type, object-id	severity, problems, trend	name, time, text, type, object-id	severity, problems, trend	name, time, text, type, object-id	severity, problems, trend	name, time, text, type, object-id	severity, problems, trend
Logs Group - state change logs	name, time, text, object-id, to state		name, time, text, object-id, to state		name, time, text, object-id, to state		name, time, text, object-id, to state		name, time, text, object-id, to state	
Logs Group - state change logs	name, time, text, object-id		name, time, text, object-id		name, time, text, object-id		name, time, text, object-id		name, time, text, object-id	

7.3 CAS component monitoring and configuration

Monitoring and configuration of CAS components is accomplished through the Simulcrypt Identification Module (SIM). SIM contains version information of the management software as well as Simulcrypt component identification, configuration, and status information. Its primary purpose is to provide uniform Simulcrypt component configuration and status information across all Simulcrypt elements of a Conditional Access System.

NOTE: This is not the only purpose of SIM as it also facilitates the transaction based C(P)SI/(P)SI interface.

If a management system is implemented, a Simulcrypt CA component needs to implement only those SIM objects, which are required for that component type by the SIMF. If a particular object is not supported by the management entity of the Simulcrypt CA component (i.e. the SNMP agent) the standard SNMP error code `noSuchName` is to be returned.

Several totals listed below, mostly concerning number of errors, refer to the total since the agent has started; other totals, mostly concerning number of streams, channels, etc., refer to current values. All `bslbf` and `uimsbf` units referred to below are in byte units.

The module can be extended by proprietary objects and groups as needed for specific DVB Simulcrypt Integrated Management Systems. SIM consists of the following object groups. All access rights can be further restricted by individual MIB views if so desired in particular implementations. In particular, a CAS device may restrict access to all its parameters to read-only mode, for allowing the NMS to access to the current status of the CAS device.

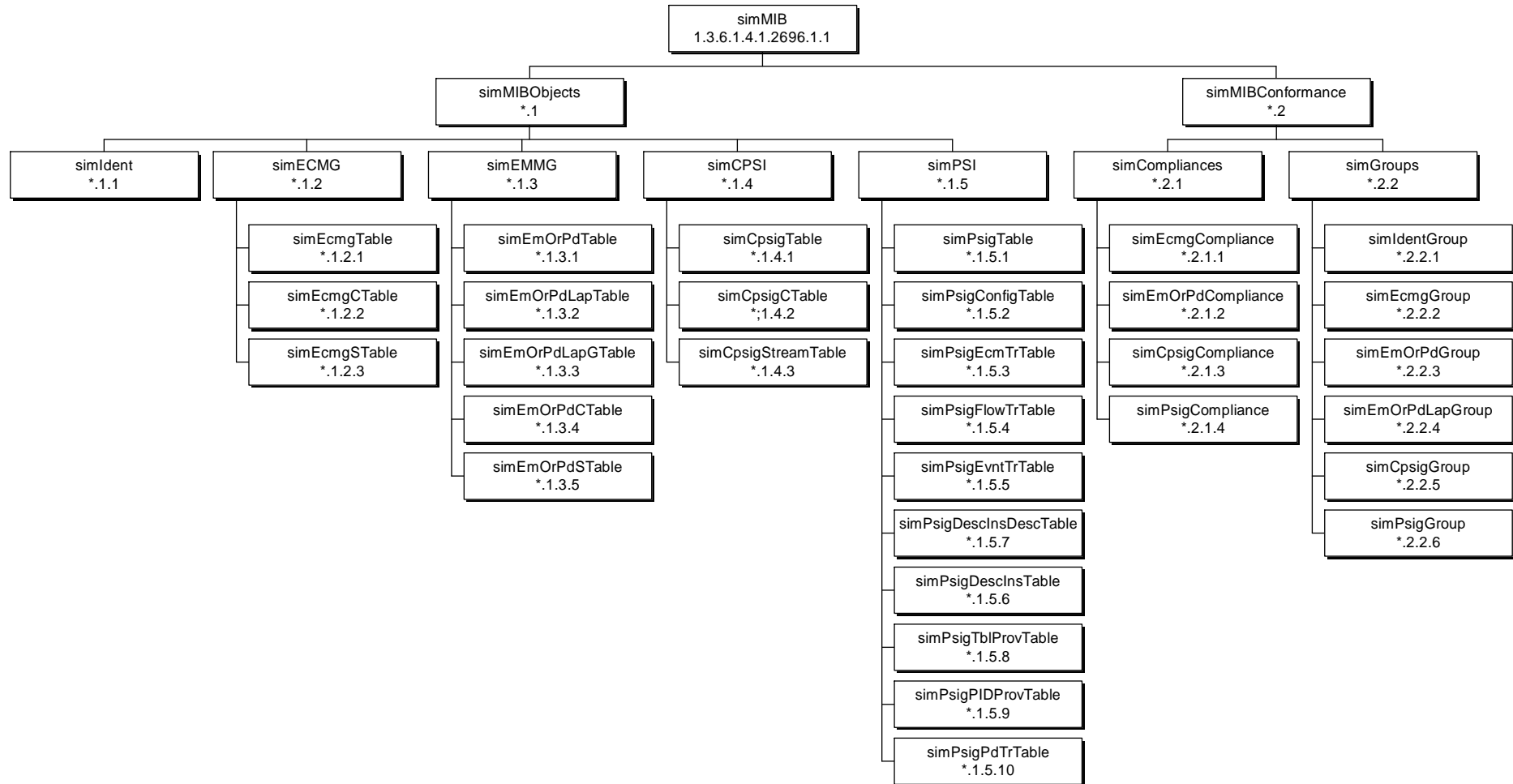


Figure 8: Simulcrypt Information Module SIM

7.3.1 Ident Group

This group is used for software configuration management of all Simulcrypt components and includes the following objects.

Table 18: CIM - Simulcrypt Identification Module

Object	Size/Description	Object Justification	Maximum Access Right
simSoftwareVersion	80 bslbf/ASCII string of software version	to facilitate software configuration management	read
simMibVersion	80 bslbf/ASCII string of MIB version	same	read
simMibPrivateVersion	80 bslbf/ASCII string of private MIB version	same	read
simAgentVersion	80 bslbf/ASCII string of agent version	same	read

7.3.2 ECM Generator Group

This group is used for configuration management and status monitoring of ECM Generators. It identifies each one of the ECM Generators by the IP Address and TCP Port Number. It associates Super_CAS_ids, ECM_channel_ids, and ECM_stream_ids with ECM Generators. It also associates status information and statistics with channels and streams. The ECM Generator Group consists of three conceptual tables.

Table 19 is the interconnection table and is used for the Head-end Network Manager to query the IP addresses and the port number to be used by an SCS to create a channel. It is indexed by a unique EcmgIndex which is an integer assigned by the ECMG agent.

Table 19: CIM - SIM ECMG Group - Interconnection Table

Object	Size/Description	Object Justification	Maximum Access Right
simEcmgIndex	2 uimsbf/unique index of the table row	allows interconnection management	read
simEcmgIpAddress	octet string of 4 octets/ IP address of the ECMG	same	read
simEcmgTcpPort	2 uimsbf/TCP port number of the ECMG	same	read
simEcmgSuCasId	4 uimsbf/Super_Cas_id	same	read
simEcmgChannels	2 uimsbf/total number of channels this ECMG is currently maintaining	same	read
simEcmgCwPrs	4 uimsbf/total number of CW Provisioning requests received by this ECMG	statistic	read
simEcmgErrs	4 uimsbf/total number of communications errors for this ECMG	same	read
simEcmgTargetCpsig	4 uimsbf/index into the C(P)SIG table identifying the C(P)SIG associated with this ECMG	interconnection management	read
simEcmgCaMib	blsb/pointer to a provider proprietary Ecmg MIB (like ifSpecific in the interfaces group of MIB II)	enables extension of ECMG MIB by proprietary modules	read

The second ECMG table (table 20) is used for monitoring channel information. It is indexed by the EcmgIndex and the ChannelId.

Table 20: CIM - SIM ECMG Group - Channel Table

Object	Size/Description	Object Justification	Maximum Access Right
simEcmgIndex	2 uimsbf/unique index of the table row	enables head-end or CAS network manager to monitor the status of individual channels and streams	read
simEcmgChannelId	2 uimsbf/identifier of a channel	same	read
simEcmgScslpAddress	octet string of 4 octets/IP address of the SCS	same	read
simEcmgScsTcpPort	2 uimsbf/TCP port number of the SCS	same	read
simEcmgCStreams	2 uimsbf/total number of streams this ECMG is currently maintaining on this channel	same	read
simEcmgCCwPrs	4 uimsbf/total number of CW Provisioning requests this ECM has received on this channel	same	read
simEcmgCErrs	4 uimsbf/total number of error messages on this channel	same	read
simEcmgCSuCasId	4 uimsbf/Super_Cas_ID	defines the CAS system and sub-system	read
simSection_TSpkt_flag	enumerated Section or TSPacket	defines the format of datagrams	read
simAC_delay_start	2 tcimsbf/AC_delay_start in ms	gives the AC_delay_start parameter value	read
simAC_delay_stop	2 tcimsbf/AC_delay_stop in ms	gives the AC_delay_stop parameter value	read
simDelay_start	2 tcimsbf/Delay_start in ms	gives the Delay_start parameter value	read
simDelay_stop	2 tcimsbf/Delay_stop in ms	gives the Delay_stop parameter value	read
simTransition_delay_start	2 tcimsbf/Transition_delay_start in ms	gives the Transition_delay_start parameter value	read
simTransition_delay_stop	2 tcimsbf/Transition_delay_stop in ms	gives the Transition_delay_stop parameter value	read
simECM_rep_period	2 uimsbf/ECM_rep_period in ms	gives the repetition period for ECM	read
simMax_streams	2 uimsbf/max_streams	gives the max number of streams supported by ECMG	read
simMin_CP_duration	2 uimsbf/min_CP_duration in n x 100 ms	gives the min crypto-period duration needed by ECMG	read
simLead_CW	1 uimsbf/lead_CW	gives the number of CW the ECMG needs in advance	read
simCW_per_msg	1 uimsbf/CW_per_msg	gives the number of CW the ECMG needs at each ECM request	read
simMax_comp_time	2 uimsbf max_comp_time in ms	gives the max delay needed by ECMG when computing an ECM	read

Table 21 is used for monitoring stream information. It is indexed by the EcmgIndex, the ChannelId, and the StreamId.

Table 21: CIM - SIM ECMG Group - Stream Table

Object	Size/Description	Object Justification	Maximum Access Right
simEcmgIndex	2 uimsbf/unique index of the table row	enables head-end or CAS network manager to monitor the status of individual channels and streams	read
simEcmgChannelId	2 uimsbf/identifier of a channel	same	read
simEcmgStreamId	2 uimsbf/identifier of a stream	same	read
simEcmgEcmlId	2 uimsbf/identifier of ECM stream	assigned by the head-end to uniquely identify each ECM stream	
simEcmgSLastCp	4 uimsbf/last crypto period processed for this stream	enables head-end or CAS network manager to monitor the status of individual channels and streams	read
simEcmgSCwPrs	4 uimsbf/total number of CW Provisioning requests this ECMG has received on this stream	same	read
simEcmgSErrs	4 uimsbf/total number of error messages for this ECMG on this Stream	same	read

7.3.3 EMMG/PDG Group

This group is used for management of EMM/PD Generators. It identifies each one of the EMM/PD Generators by the IP Address and TCP/UDP Port Number. It also associates client_ids, data_channel_ids, and data_stream_ids with EMM/PD Generators. It also associates status information and statistics with streams. The EMMG/PDG Generator Group consists of five conceptual tables.

Table 22 is used for information relevant to EMMG/PDG and is indexed by a unique EmOrPdIndex, which is assigned by the EMMG/PDG agent.

Table 22: CIM - SIM EMMG/PDG Group - Channel Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simEmOrPdIndex	2 uimsbf/unique index of the EMMG/PDG	identifies the EMMG/PDG	read
simEmOrPdLapIndex	2 uimsbf/logical global unique identifier	enables the association of a global logical identifier with a mux IP address and port	read
simEmOrPdChannelId	2 uimsbf/identifier of a channel	identifies channel and enables monitoring of it	read
simEmOrPdCommType	enumerated/TCP or UDP	defines the communications type for the channel (TCP or UDP)	read-create
simEmOrPdCIPAddress	octet string of 4 octets/IP address	used with mux Ip address and port number to identify the TCP connection used for access of MIB II	read
simEmOrPdCPort	2 uimsbf/TCP/UDP port number	same	read
simEmOrPdCErrs	2 uimsbf/total number of communications errors on this channel	statistic	read
SimEmOrPdFormat	Enumerated Section or TSPacket	defines the format of datagrams	read/create

Table 23 is used for configuration of EMMG/PDG. It is uniquely indexed by the EmOrPdLapIndex, which is a globally assigned quantity (with respect to the head-end) and associates globally assigned Logical Access Points (LAPs) with mux ports.

Table 23: CIM - SIM EMMG/PDG Group - Logical Access Point Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simEmOrPdLapIndex	2 uimsbf/global unique Logical Access Point (LAP) identifier	enables the association of a global logical identifier with a mux IP address and port	read-create
simEmOrPdAdminState	enumerated (ITU-T Recommendation X.734 [6])/the administrative state of a table row	used to control concurrent access to writeable objects as defined in ITU-T Recommendation X.734 [6]	read-create
simEmOrPdLapCommType	enumerated/TCP or UDP	defines communications type	read-create
simEmOrPdLapMuxIpAddress	octet string of 4 octets/IP address of the MUX	identifies the mux's IP address	read-create
simEmOrPdLapMuxPort	2 uimsbf/TCP/UDP port of the MUX	identifies the mux's port	read-create
simEmOrPdLapStatus	enumerated/row status of the entry, RowStatus (SNMP)	used to manage the creation/deletion of rows in a table	read-create
simEmOrPdLapMuxUlpAddress	string of 4 octets/IP address	identifies the mux's UDP IP address	read-create
simEmOrPdLapMuxUPort	2 uimsbf/UDP port number	identifies the mux's UDP port	read-create

Table 24 is also used for configuration of EMMG/PDG. It associates LAP Groups and LAPs and is uniquely indexed by the EmOrPdLapGroup, and EmOrPdLapIndex. This association permits a manager to configure an EMMG/PDG to service multiple multiplexers and/or multiple multiplexer ports.

Table 24: CIM - SIM EMMG/PDG Group - Logical Access Point Group Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simEmOrPdLapGroup	2 uimsbf/not-unique identifier associating EMMGs to EMM populations	enables the establishment of multiple channels by the EMMG based on LapGroup	read-create
simEmOrPdLapIndex	2 uimsbf/global unique Logical Access Point (LAP) identifier	enables the association of a global logical identifier with a mux IP address and port	read-create
simEmOrPdLapGAdminState	enumerated (ITU-T Recommendation X.734 [6])/the administrative state of a table row	used to control concurrent access to writeable objects as defined in ITU-T Recommendation X.734 [6]	read-create
simEmOrPdLapGStatus	enumerated/row status of the entry, RowStatus (SNMP)	used to manage the creation/deletion of rows in a table	read-create

Table 25 is used for channel related monitoring information. Table 23 is indexed by the EmOrPdIndex, EmOrPdLapIndex, and channel_id.

Table 25: CIM - SIM EMMG/PDG Group - Channel Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simEmOrPdIndex	2 uimsbf/unique index of the EMMG/PDG	identifies the EMMG/PDG	read
simEmOrPdLapIndex	2 uimsbf/logical global unique identifier	enables the association of a global logical identifier with a mux IP address and port	read
simEmOrPdChannelId	2 uimsbf/identifier of a channel	identifies channel and enables monitoring of it	read
simEmOrPdCommType	enumerated/TCP or UDP	defines the communications type for the channel (TCP or UDP)	read-create
simEmOrPdCipAddress	octet string of 4 octets/IP address	used with mux TCP/IP address and port number to identify the TCP connection used for access of MIB II.	read
simEmOrPdCPort	2 uimsbf/TCP/UDP port number	same	read
simEmOrPdCErrs	2 uimsbf/total number of communications errors on this channel	statistic	read
SimEmOrPdFormat	Enumerated Section or TSPacket	defines the format of datagrams	read/create
SimEmOrPdFormat	Enumerated Section or TSPacket	defines the format of datagrams	read
simEmOrPdCUdplp Address	string of 4 octets/IP address	used with MUX UDP Ip address and port number to identify the UDP link	read
simEmOrPdCUdpPort	2 uimsbf/UDP port number	same	read

Table 26 contains stream related information. It is indexed by the simEmOrPdIndex, simEmOrPDLapIndex, and simEmOrPdDataId.

Table 26: CIM - SIM EMMG/PDG Group - Stream Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simEmOrPdIndex	2 uimsbf/unique index of the EMMG/PDG	identifies the EMMG/PDG	read
simEmOrPdLapIndex	2 uimsbf/logical global unique identifier	enables the association of a global logical identifier with a mux IP address and port	read-create
simEmOrPdDataId	2 uimsbf/unique EMM or PD stream identifier	assigned by the head-end to uniquely identify the EMM or PD stream	read-create
simEmOrPdChannelId	2 uimsbf/identifier of a channel	enables head-end or CAS network manager to monitor the status	read
simEmOrPdBwidth	4 uimsbf/bandwidth	maximum bandwidth allocated for this EMM/Private Data stream	read-create
simEmOrPdStreamId	2 uimsbf/identifier of a stream	enables head-end or CAS network manager to monitor the status	read
simEmOrPdSErrs	4 uimsbf/total number of communications errors on this stream	statistic	read
simEmOrPdSBytes	4 uimsbf/total number of bytes sent by the EMMG/PDG on this stream	statistic	Read
simEmOrPdSReq Bwidth	4 uimsf/bandwidth	bandwidth requested by the EMMG/PDG for this stream	read-create

7.3.4 C(P)SIG Group

This group is used if the (P)SIG Group is not implemented and the stream/channel based protocol between the Custom (P)SIG and (P)SIG is implemented.

The C(P)SIG Group enables management of some aspects of the interaction between the Custom (P)SI Generators (C(P)SIG) and the (P)SI generator. It identifies each one of the C(P)SI Generators by the IP Address and TCP Port Number. It also associates Super_CAS_ids, Custom_channel_ids, Custom_stream_ids with C(P)SI Generators. It also associates status information and statistics with streams. The C(P)SI Generator Group consists of three conceptual tables.

Table 27 specifies C(P)SIG related information, which is posted by the entity hosting or representing the C(P)SIG. A CA/Head-end manager can read this table to communicate TCP/IP information to the (P)SIG, which will establish the TCP connection with the C(P)SIG. Table 25 is indexed by a unique CpsigIndex, which is assigned by the CPSIG agent.

Table 27: CIM - SIM C(P)SIG Group - C(P)SIG Table

Object	Size/Description	Object Justification	Maximum Access Right
simCpsigIndex	2 uimbsf/unique index of the table row	provides a unique identification of a row	read
simCpsigSuperCasId	4 uimbsf/Super_CAS_id	client ID	read
simCpsigErrs	4 uimbsf/total number of communications errors for this C(P)SIG	statistic	read
simCpsigChannels	4 uimbsf/total number of channels currently maintained by this C(P)SIG	statistic	read
simCpsigCpsigIpAddress	octet string of 4 octets/IP address of the C(P)SIG	the IP address of the C(P)SIG	read
simCpsigCpsigPort	4 uimbsf/TCP port of the C(P)SIG	the port number of the C(P)SIG	read
simCpsigCaMib	bslbf/pointer to CA provider proprietary MIB, OBJECT IDENTIFIER (SNMP)	enables custom MIB extensions	read

Table 28 is used for channel related monitoring information. The table is indexed by the CpsigIndex and custom_channel_id.

Table 28: CIM - SIM C(P)SIG Group - Channel Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simCpsigIndex	2 uimbsf/unique index of the C(P)SIG	refers to the index in the first table and identifies the C(P)SIG	read
simCpsigChannelId	2 uimbsf/identifier of a channel	identifies channel and enables monitoring of it	read
simCpsigPsigIpAddress	octet string of 4 octets/IP address of the (P)SIG	used with the port number and C(P)SIG IP address and port number can identify the TCP connection used for access of MIB II	read
simCpsigPsigPort	2 uimbsf/TCP port number of the (P)SIG	same	read
simCpsigCErrs	4 uimbsf/total number of communications errors on this channel	statistic	read
simCpsigCTstrms	4 uimbsf/total number of "transport" streams on this channel	statistic	read
simCpsigCSstrms	4 uimbsf/total number of "service" streams on this channel	statistic	read

Table 29 contains "transport" stream related information. It is indexed by the CpsigIndex, custom_channel_id, and custom_stream_id.

Table 29: CIM - SIM C(P)SIG Group - Stream Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simCpsigIndex	2 uimbsf/unique index of the C(P)SIG	refers to the index in the first table and identifies the C(P)SIG	read
simCpsigChannelId	2 uimbsf/identifier of a channel	enables head-end or CAS network manager to monitor the status of streams on a channel	read
simCpsigStreamId	2 uimbsf/identifier of a stream	same	read
simCpsigStreamTStreamId	2 uimbsf/identifier of a stream	associates a transport stream with the stream	read
simCpsigStreamNid	2 uimbsf/network identifier	associates a network with this stream	read
simCpsigStreamOnid	2 uimbsf/original network identifier of this stream	associates an original network id with the stream	read
simCpsigStreamMaxCompTime	2 uimbsf/maximum time needed to process trigger by C(P)SIG	enables (P)SIG to estimate the time between its triggering and descriptor insertion by the C(P)SIG	read
simCpsigStreamTriggerEnable	2 uimbsf/trigger type definition	identifies which trigger types the C(P)SIG wants	read
simCpsigStreamLastTrigger	2 uimbsf/trigger type	identifies the type of the last trigger processed	read
simCpsigStreamLastEventId	2 uimbsf/event identifier	identifies the last event for which a trigger has been processed	read
simCpsigStreamLastServiceId	2 uimbsf/the service identifier	identifies the service related to the last ECM related trigger	read
simCpsigStreamLastEsId	2 uimbsf/the elementary stream identifier	identifies the elementary stream of the last trigger if the ECMs are to be applied component wide only and not service wide	read
simCpsigStreamLastEcmPid	2 uimbsf/the ECM PID	identifies the ECM PID of the last ECM related trigger	read
simCpsigStreamErrs	4 uimbsf/total number of communications errors on this stream	statistic	read
simCpsigStreamBytes	4 uimbsf/total number of bytes sent by the CPSIG on this stream	statistic	read

7.3.5 Conformance Requirements

Table 30 summarizes the conformance requirements for management entities implementing the Simulcrypt Identification Module (SIM), by group.

Table 30: CIM - SIM Conformance Requirements

Common Information Model - CIM Simulcrypt Identification Module Group	Management Entity Hosting or Representing an ECMG		Management Entity Hosting or Representing an EMMG		Management Entity Hosting or Representing a PDG		Management Entity Hosting or Representing a (P)SIG		Management Entity Hosting or Representing a C(P)SIG	
	mndt	optnl	mndt	optnl	mndt	optnl	mndt	optnl	mndt	optnl
Ident Group	X		X		X		X		X	
ECMG Group	X									
EMMG/PDG Group (without LAPG table)			X		X					
EMMG/PDG Group (LAPG Table)				X		X				
PSIG Group							X			
CPSI Group									X	

8 C(P)SIG \Leftrightarrow (P)SIG interface

8.1 Overview and Scope

This clause defines the interface that allows one or more CA Systems (CASs) to insert private data into standard MPEG-2 PSI and DVB SI tables that are generated by the head-end system.

The following clauses detail an application protocol model for the C(P)SIG \Leftrightarrow (P)SIG interface and two implementation-oriented protocols (connection-oriented protocol and SIMF-based protocol) based on this model.

Clause 8.2 presents the model of application protocol defined to support the C(P)SIG \Leftrightarrow (P)SIG interface.

Clause 8.3 specifies the C(P)SIG \Leftrightarrow (P)SIG protocol defined on the basis of command/response messages; it includes the state machines definition and the messages description.

Clause 8.4 specifies the C(P)SIG \Leftrightarrow (P)SIG protocol defined in the network management environment; it includes the description of dedicated data structure (MIB) and the definition of rules of operation.

The head-end and each CAS shall comply with the requirements presented in clauses E.1 and E.2 respectively, as well as all specifications documented in the above-described clauses.

As defined in the clause 4 and shown in figure 9, the logical components involved are as follows:

- **PSI Generator (PSIG):** the head-end process(es) responsible for generating MPEG-2 PSI (Program Specific Information) tables;
- **SI Generator (SIG):** the head-end process(es) responsible for generating DVB SI (Service Information) tables;

NOTE: The NIT (Network Information Table) is considered a DVB SI table.

- **Custom PSI Generator (CPSIG):** the CA System (CAS) process(es) responsible for generating CAS-specific private data for insertion in selected MPEG-2 PSI tables;
- **Custom SI Generator (CSIG):** the CAS process(es) responsible for generating CAS-specific private data for insertion in selected DVB SI tables;
- the generic term **(P)SIG** refers to a head-end process that serves as a PSIG, an SIG, or both (PSISIG). If the head-end supports the C(P)SIG \Leftrightarrow (P)SIG interface, it shall include one or more (P)SIG;

- the generic term **C(P)SIG** refers to a head-end process that serves as a CPSIG, a CSIG, or both (CPSISIG). Each CAS may (optionally) include one or more C(P)SIG.

This clause defines the interface between the C(P)SIG and the (P)SIG.

Table 31: Values of (P)SIG_type parameter

(P)SIG_type values	C(P)SIG or (P)SIG processes
0x01	PSIG
0x02	SIG
0x03	PSISIG
0x04	CPSIG
0x08	CSIG
0x0C	CPSISIG
other values	DVB reserved

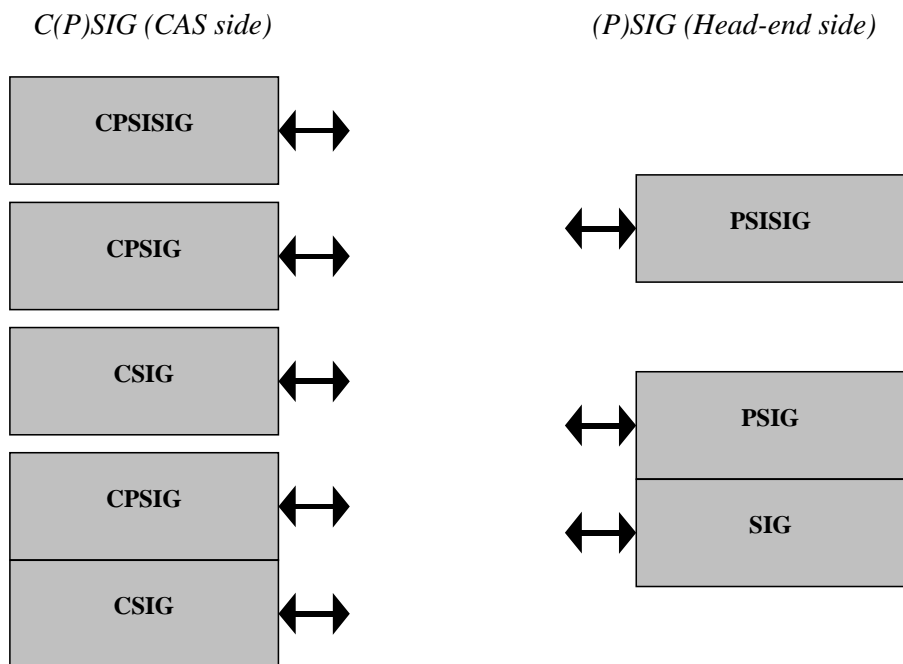


Figure 9: Possible logical architectures of C(P)SIG and (P)SIG

8.1.1 Note on commercial agreements

Certain aspects of the C(P)SIG \Leftrightarrow (P)SIG interface fall outside the scope of the present document, and shall be decided by commercial agreements. One such example is the set of non-mandatory PSI and SI tables generated by a (P)SIG. All such aspects of this interface are indicated in the text.

8.1.2 Note on the PDG \Leftrightarrow MUX Interface

A CAS may also create private data in the form of private sections that comply with MPEG-2 and DVB section syntax. The PDG \Leftrightarrow MUX interface in clause 6 should be used to insert such private sections.

8.2 Application protocol model

8.2.1 Overview of the C(P)SIG \Leftrightarrow (P)SIG Application Protocol

The C(P)SIG \Leftrightarrow (P)SIG Application Protocol includes and defines the following five transaction types:

- **Trigger Transactions:** Triggers are asynchronous events from the (P)SIG to the C(P)SIG. The (P)SIG first informs the C(P)SIG about the types of triggers it supports. The C(P)SIG then informs the (P)SIG which types of triggers it wishes to receive. The (P)SIG then communicates these triggers to the C(P)SIG as the corresponding events occur;
- **(P)SI Table Provisioning Transactions:** The C(P)SIG requests (P)SI tables from the (P)SIG and the (P)SIG communicates those to the C(P)SIG;
- **C(P)SIG Descriptor Insert Transactions:** The C(P)SIG requests that the (P)SIG inserts (P)SI descriptors into the transport stream. The (P)SIG informs the C(P)SIG about the success of the insertion;
- **Service Change Transactions:** The (P)SIG informs the C(P)SIG about a modification in service existence in a transport stream (addition of a service to a TS or deletion of a service from a TS);
- **PID Provisioning Transactions:** The C(P)SIG requests from the (P)SIG the PID value assigned by the head-end to a stream (ECM, EMM or private data) identified by a unique identifier which includes the type of the stream, the identifier of the CA system and the CA sub-system the stream belongs to, and an unique stream number.

In the following clauses configurations, topologies and the five different transaction types are defined. The transaction type specifications include only the definitions of the data exchanged by the components. The actual data exchange mechanisms are transport dependent and are fully defined in the following two clauses.

8.2.2 Configurations and Topologies

Figure 10 illustrates the relevant components in a head-end system with a single CA System (CAS) and in a head-end system with multiple CASs. In both diagrams, the *M-to-N* and *I-to-N* notation reflects logical associations, not process interconnections.

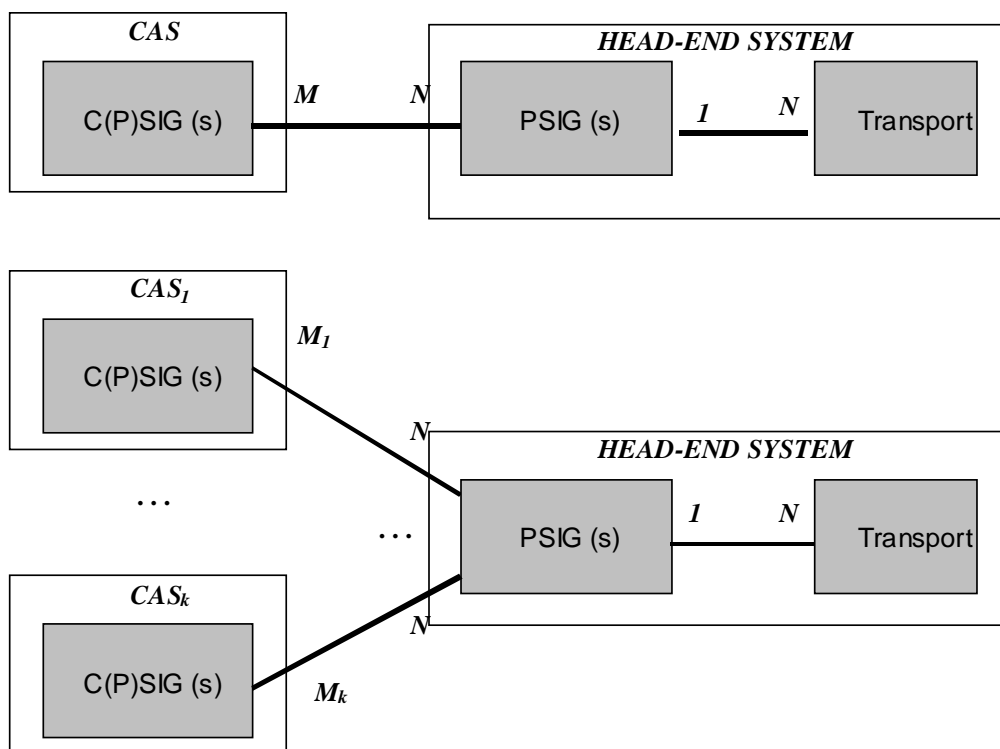


Figure 10: Head-end system with single CAS (top) and with multiple CASs (bottom)

The head-end includes the following components:

- one or more transport streams (TSs) that conform to MPEG-2 and DVB specifications. The C(P)SIG \Leftrightarrow (P)SIG Interface views each TS, and its data streams, as logical entities. As such, each TS is uniquely identified, per DVB SI, by the triple {original_network_id, network_id, transport_stream_id}. The C(P)SIG \Leftrightarrow (P)SIG interface does not deal with physical components (namely MUXes) that generate TSs;
- one or more (P)SIG components. Each (P)SIG is uniquely identified by its IP address and TCP port number;
- connections between the (P)SIGs and the TSs. The relationship between (P)SIG and TS is *1-to-N*: each (P)SIG generates standard PSI and/or SI tables for one or more TSs and no TS is served by more than one (P)SIG.

The head-end shall supply either a PSISIG or a {PSIG+SIG} to serve each TS.

The present document does not define the physical connection between (P)SIG and TS; it is a matter of head-end implementation.

Each CAS that supports the C(P)SIG \Leftrightarrow (P)SIG Interface includes the following components:

- one or more C(P)SIG components. Each C(P)SIG is uniquely identified by a custom_CAS_id, as well as its IP address and TCP port number.

The relationship between the C(P)SIG(s) of a CAS, and the (P)SIG(s) of the head-end, is *M-to-N*. Given also the *1-to-N* relationship of (P)SIG to TS, this means:

- each C(P)SIG may generate CAS-specific private data for zero or more TSs;
- any TS may be served by zero or more C(P)SIGs, as long as said C(P)SIGs and TS are connected to the same (P)SIG.

The components of each CAS are logically independent of each other. Any TS may be served by zero or more C(P)SIGs from one or more CASs, as long as said C(P)SIGs and TS are connected to the same (P)SIG.

8.2.3 Trigger Transaction Type

The trigger is the means by which a (P)SIG component indicates to a C(P)SIG that a particular event is going to take place.

DVB Simulcrypt has identified the following trigger types:

- Following DVB-event;
- Future DVB-event;
- ECM stream setup;
- Access criteria change;
- ECM stream closure;
- Flow PID change;
- PD stream event.

The **Following DVB-event** trigger indicates that a new event is going to be broadcast on a given service.

The **Future DVB-event** trigger indicates that the head-end has at its disposal some information concerning a future event. The moment at which this message is sent is not specified by DVB.

The **ECM stream setup** trigger indicates that a new ECM stream is going to be opened.

The **Access criteria change** trigger indicates that the access criteria of an existing ECM stream is going to be changed.

The **ECM stream closure** trigger indicates that an ECM stream is due for closure.

The **Flow PID change** trigger indicates that the PID value of an existing ECM, EMM or private data stream is going to be changed.

The **PD stream event** trigger indicates that an event (creation of the stream, addition of an already existing PD stream to a new service, modification of the possibly associated CAS private data, etc.) is going to occur for one particular PD stream.

With the exception of the **Future DVB-event** trigger, the time at which the trigger message is sent with respect to the corresponding event is computed by the (P)SIG, by subtracting the **max_comp_time** value from the time at which the corresponding event takes place. This lets time for the C(P)SIG to reply with descriptor_insert messages if needed, and control the precise timing of the insertion of descriptors.

For the head-end, the support of the **Future DVB-Event** is optional.

During the channel setup, the (P)SIG indicates what kind of triggers it is able to generate. The C(P)SIG has the possibility to subscribe to some trigger types per service. A trigger subscription is on a service basis, meaning, that a C(P)SIG can subscribe to different triggers depending on the services: such a trigger subscription can be modified at any time.

The (P)SIG communicates to the C(P)SIG the types of triggers its supports as an octet string of length 4 (**trigger_list**) encoded as bit mask with the meaning given in table 32 if set:

Table 32: Trigger types

trigger cause	trigger_list bit #	trigger type
DVB reserved	none	0x00000000
following event	0 (LSB)	0x00000001
future event	1	0x00000002
ECM stream setup	2	0x00000004
access criteria change	3	0x00000008
ECM stream closure	4	0x00000010
flow PID change	5	0x00000020
PD stream event	6	0x00000040
DVB reserved	7 to 15	0x00000080 to 0x00008000 (powers of 2 only)
user defined	16 to 31 (MSB)	0x00010000 to 0x80000000 (powers of 2 only)
combination of any previous	any combination	any value from 0x00000000 to 0xFFFFFFFF

The C(P)SIG communicates to the (P)SIG which triggers it wishes to receive using the same mask.

The trigger information communicated by the (P)SIG to the C(P)SIG includes the following:

- original network identifier as defined by DVB SI (original_network_id);
- network identifier as defined by DVB SI (network_id);
- transport identifier as defined by DVB SI (transport_stream_id);
- service identifier as defined by DVB SI or the MPEG PAT program_number (service_id);
- trigger type coded the same way as the trigger mask defined above;
- either event related data, ECM related data, flow PID change related data or PD related data depending on the trigger type.

Event related data is defined as follows:

- event identifier as defined in EN 300 468 [1] (event_id);
- event duration in hours, minutes, seconds coded as six 4-bit BCD digits;
- start time of the event in Modified Julian Date (MJD) and Universal Time, Coordinated (UTC) formats. It is contained in an octet string of length 5. The MJD is coded in the 16 LSBs followed by six 4-bit BCD digits representing UTC;
- private data which correspond to the event private data.

ECM related data is coded as follows:

- elementary stream identifier of the elementary stream to which the ECM stream is attached;
- ECM PID;
- ECM Identifier which corresponds to the ECM channel identifier, the ECM stream identifier, and the Super_CAS identifier;
- access criteria.

Flow-PID change related data is coded as follows:

- flow-type;
- flow-super-CAS-id;
- flow-id;
- flow-PID.

PD related data is coded as follows:

- elementary stream identifier of the PD stream in the PMT of the related service;
- PID allocated to the PD stream;
- Private Data stream identifier which corresponds to the Private Data client_id, the Private Data channel identifier and the PD stream identifier;
- PD stream type in the PMT;
- possible CAS proprietary data.

8.2.4 Table Provisioning Transaction Type

The C(P)SIG communicates to the (P)SIG the following information uniquely identifying in a given transport-stream the table the C(P)SIG is requesting, according to table 33.

Original network identifier and transport identifier (original_network_id and transport_stream_id as defined by DVB SI) identifying the transport stream carrying the requested table.

Table identifier:

- original network identifier as defined by DVB SI (original_network_id);
- network identifier as defined by DVB SI (network_id);
- transport identifier as defined by DVB SI (transport_stream_id);
- service identifier as defined by DVB SI (service_id) or the MPEG PAT program_number;
- bouquet identifier as defined by DVB SI (bouquet_id);
- event identifier as defined by DVB SI (event_id);
- segment number in an EIT schedule as defined by DVB SI.

Table 33: Table Identifier coding and required parameters

Table	Table_id	Required parameters
all tables: identification of the transport stream carrying the requested table	-	original_network_id transport_stream_id
PAT	0x00	-
CAT	0x01	-
PMT	0x02	service_id (MPEG-2 program number)
NIT actual network	0x40	-
NIT other network	0x41	network_id _{other}
BAT	0x4A	bouquet_id
SDT actual TS	0x42	-
SDT other TS	0x46	original_network_id _{other} transport_stream_id _{other}
EIT p/f actual TS	0x4E	service_id
EIT p/f other TS	0x4F	original_network_id _{other} transport_stream_id _{other} service_id
EIT schedule actual TS (whole sub-table)	0x50 - 0x5F	service_id
EIT schedule actual TS (single segment)	0x50 - 0x5F	service_id segment_number
EIT schedule actual TS (single event only)	0x50	service_id event_id
EIT schedule other TS (whole sub-table)	0x60 - 0x6F	original_network_id _{other} transport_stream_id _{other} service_id
EIT schedule other TS (single segment)	0x60 - 0x6F	original_network_id _{other} transport_stream_id _{other} service_id segment_number
EIT schedule other TS (single event only)	0x60	original_network_id _{other} transport_stream_id _{other} service_id event_id

The C(P)SIG requests either complete PSI tables or SI sub-tables (see EN 300 468 [1]). As EIT Schedule sub-tables may be very large, the protocol optionally allows the C(P)SIG to request only the part of the EIT Schedule sub-table that concerns a specific event or a specific segment. In case the request concerns a specific event_id, the value of the table_id field shall be set to 0x50 for the events pertaining to the current TS and 0x60 for the events pertaining to an EIT other.

The table requested is supplied by the (P)SIG to the C(P)SIG if available. The communications mechanism is transport dependent and specified in the next two clauses.

8.2.5 Descriptor Insertion Transaction Type

The C(P)SIG communicates to the (P)SIG the descriptors that the C(P)SIG wants to be inserted in the PSI/SI tables of a given transport stream. This includes all the information that the (P)SIG needs to know where and when to insert the descriptors. The information depends on the type of descriptor inserted and is specified below:

- original network identifier and transport stream identifier (original_network_id and transport_stream_id as defined by DVB SI) identifying the transport stream carrying the table where descriptors are to be inserted;
- trigger identifier if related to a previously transmitted trigger;
- insertion delay type which can be either *immediate* (i.e. no delay) or *synchronized* with the cause of a trigger if the trigger is identified;

- insertion delay: this parameter is significant only if insertion delay type is *synchronized* in seconds; the delay value can be positive in which case it indicates that the synchronization is to happen the delay time after the trigger cause, or negative in which case it indicates that the synchronization is to happen the delay time before the cause;
- location identifier, which identifies the table and descriptor, loop if applicable in which the (P)SIG is to insert the set of descriptors;
- original network identifier as defined by DVB SI (original_network_id);
- network identifier as defined by DVB SI (network_id);
- transport identifier as defined by DVB SI (transport_stream_id);
- service identifier as defined by DVB SI or the MPEG PAT program_number (service_id);
- bouquet identifier as defined by DVB SI (bouquet_id);
- event identifier as defined by DVB SI (event_id);
- elementary stream identifier either of the elementary stream to which the ECM stream is attached or of the PD stream in the PMT;
- private data specifier as specified by EN 300 468 [1] and ETR 162 [i.2];
- descriptor to be inserted.

The private_data_specifier parameter enables the C(P)SIG to request the insertion of the set of descriptors within the scope of a private_data_specifier descriptor (as defined in EN 300 468 [1]), if supported by the head-end.

The insertion_delay_type and insertion_delay parameters provide synchronization information. They indicate to the (P)SIG when to insert the set of descriptors with respect to the ECM stream modification time or the event start time. The insertion_delay parameter indicates the amount of time between the insertion of the table and the insertion of the ECMs in the transport stream. If it is positive, it means that the insertion of the table in the transport stream shall occur after the start of the ECM broadcast. If negative, it means that the modified PSI/SI table shall be broadcast ahead of the ECM modification time in the transport stream.

Table 34 indicates, for each MPEG-2 PSI and DVB SI table, which parameters are needed.

Table 34: Location Identifiers and Required Parameters

Table	Location_id value	Required location parameters
all tables: identification of the transport stream carrying the targeted table	-	original_network_id transport_stream_id
CAT	0x01	—
PMT 1 st loop	0x02	service_id (MPEG-2 program number)
PMT 2 nd loop	0x03	service_id (MPEG-2 program number) ES_id
NIT 1 st loop - actual network	0x04	—
NIT 2 nd loop - actual network	0x05	original_network_id _(2nd loop) transport_stream_id _(2nd loop)
NIT 1 st loop - other network	0x06	network_id _{other}
NIT 2 nd loop - other network	0x07	network_id _{other} original_network_id _(2nd loop) transport_stream_id _(2nd loop)
BAT 1 st loop	0x08	bouquet_id
BAT 2 nd loop	0x09	bouquet_id original_network_id _(2nd loop) transport_stream_id _(2nd loop)
SDT (actual TS)	0x0A	service_id
SDT (other TS)	0x0B	original_network_id _{other} transport_stream_id _{other} service_id
EIT present/following (actual TS)	0x0C	service_id event_id
EIT present/following (other TS)	0x0D	original_network_id _{other} transport_stream_id _{other} service_id event_id
EIT schedule (actual TS)	0x0E	service_id event_id
EIT schedule (other TS)	0x0F	original_network_id _{other} transport_stream_id _{other} service_id event_id

The descriptor insertion "location" (PSI/SI table and descriptor loop, if applicable) is unambiguously qualified by the location_id and the location parameters given above. When a given C(P)SIG provides a set of descriptors for insertion, that set replaces the previous set at the same location (in the same private_data_specifier context).

The C(P)SIG can associate the descriptor insertion with a trigger by supplying a trigger_id parameter.

8.2.6 Service Change Transaction Type

The (P)SIG informs the C(P)SIG about a modification in service existence in a transport stream (addition of a service to a transport stream or deletion of a service from a transport stream). More sophisticated functions such as service_id change or the move of a service from one transport stream to another transport stream can be achieved by combining these two basic addition and deletion functions. The (P)SIG can give as well the scheduled day and time of the service change in the transport stream.

8.2.7 Flow PID Provisioning Transaction Type

The PID provision functionality is available only in the C(P)SIG \Leftrightarrow (P)SIG protocol. As used in this clause, the word "flow" concerns an ECM stream, an EMM stream or a private data stream in a particular transport stream. A (P)SIG can provide a C(P)SIG with the PID of a flow by one of these four ways:

- by explicit request from the C(P)SIG to the (P)SIG: the flow PID provisioning functionality described in this clause;
- by triggering the C(P)SIG when a change of PID occurs in a flow;
- for an ECM stream, by triggering the C(P)SIG when a new ECM stream is created;
- for PD stream by triggering the C(P)SIG when a new event occurs for this stream.

A flow is unambiguously known by the head-end and by the CAS by using its unique identifier defined as the combination of:

- the type of the flow: flow_type = ECM, EMM or private data;
- the Super CAS_id this flow belongs to: flow_super_CAS_id;
- an individual number: flow_id; for a flow_super_CAS_id and a flow_type, the flow_id shall be unique.

Such a combination identifies uniquely a flow across all the Simulcrypt protocols used in a real configuration and across all the transport streams generated by the head-end.

Depending on the type of the flow, its unique identifier is assigned by the head-end (ECM) or by the CAS (EMM, private data) when the flow is created.

When a unique identifier exists in the system, the head-end is assumed to know which flow_PID value is associated with.

The consequences are that:

- the unique identifier of an ECM flow (absolute reference) is carried in the ECMG protocol in addition to the channel_id and stream_id parameters (references relative to the current ECMG-SCS connection); it shall be assigned by the head-end and provided to the ECMG at stream setup;
- the unique identifier of an EMM/private data flow (absolute reference) is carried in the EMMG/PDG protocol in addition to the channel_id and stream_id parameters (references relative to the current EMMG/PDG-MUX connection); it shall be assigned by the EMMG/PDG and provided to the head-end at stream setup;
- in the C(P)SIG protocol, the PID provisioning functionality allows the C(P)SIG to get the flow_PID value of a particular flow identified by its unique identifier;
- in the C(P)SIG protocol, the flow-PID-change trigger functionality allows the (P)SIG to trigger the C(P)SIG when a change occurs for the flow_PID value of a particular flow identified by its unique identifier;
- in the C(P)SIG protocol, the flow-PID of an ECM stream (i.e. ECM-PID) is given among other ECM related parameters by the (P)SIG when the ECM stream is created;
- in the C(P)SIG protocol, the flow-PID of a PD stream (i.e. data-PID) is given by the (P)SIG among other PD related parameters when a PD stream event occurs being signalled by a PD stream event trigger message;
- it is assumed that a CAS can support internal links between ECMG and C(P)SIG or between EMMG/PDG and C(P)SIG to manage consistently the unique identifiers it is concerned by.

Figure 11 depicts this mechanism:

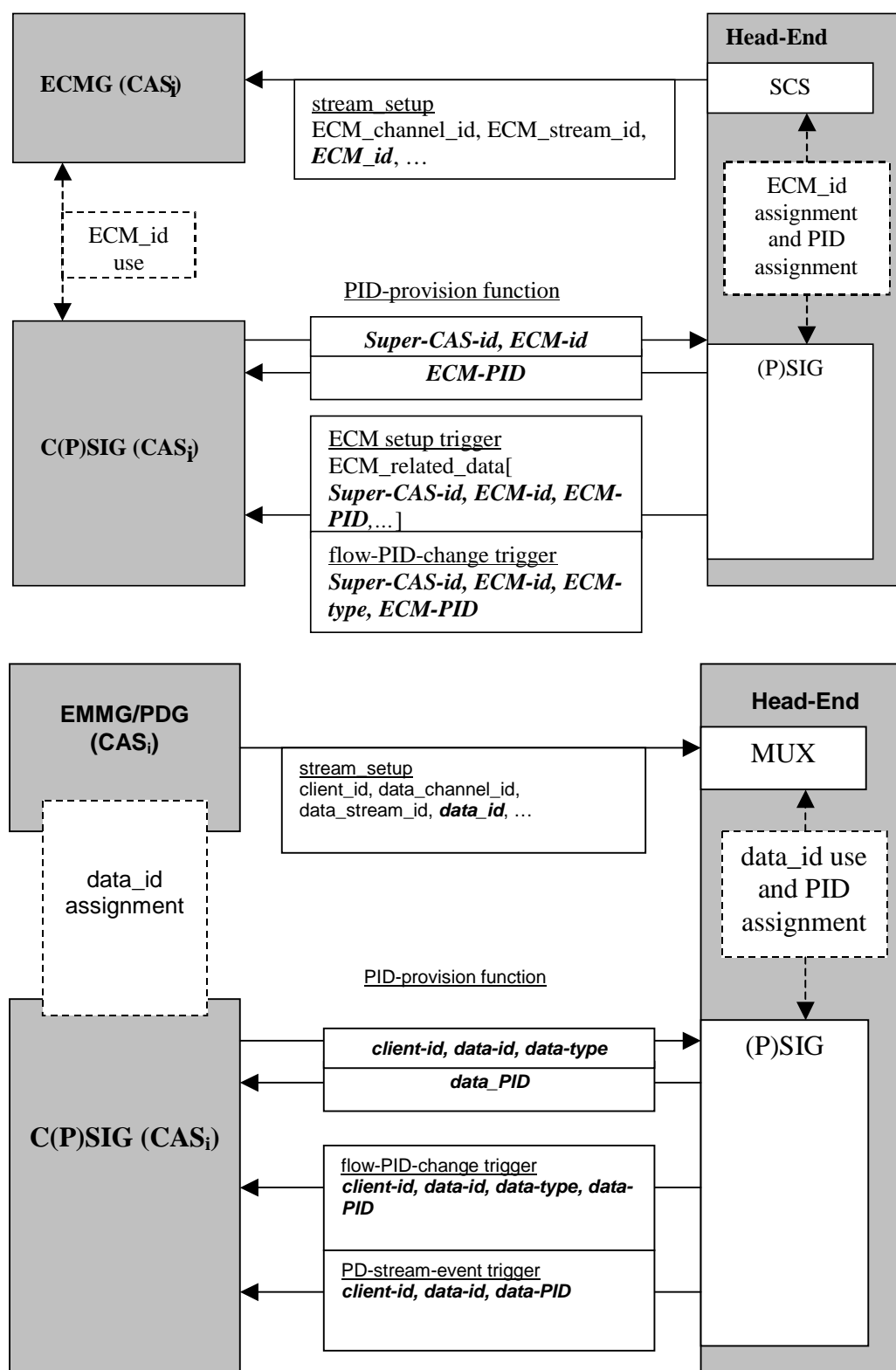
- for ECM stream (in this case ECM_id and flow_id are homonym);
- for EMM/private data stream (in this case data_id and flow_id are homonym, data_PID and flow_PID as well);
- for PD stream, in particular on PD stream event trigger: in this case data_id and flow_id are homonym, data_PID and flow_PID as well.

In the ECMG protocol, the type of the flow is implicitly ECM; the flow_super_CAS_id is the Super_CAS_id, which is implicitly in the channel_id. Hence in this protocol only the flow_id shall be given explicitly.

In the EMMG/PDG protocol, the type of the flow is given in the data_type parameter (EMM or PDG); the flow_super_CAS_id is the client_id, which is implicitly in the channel_id. Hence in this protocol only the flow_id shall be given explicitly.

In the C(P)SIG protocol:

- in the PID-provision function, the three parameters flow_type, flow_super_CAS_id, flow_id shall be given explicitly; in particular, the flow_super_CAS_id is not always the custom_CAS_id;
- in the ECM-setup trigger function, the flow_type is implicitly ECM; so both parameters flow_super_CAS_id and flow_id shall be given explicitly; in particular, the flow_super_CAS_id is not always the custom_CAS_id;
- in the flow-PID-change trigger function, the three parameters flow_type, flow_super_CAS_id and flow_id shall be given explicitly;
- in the PD stream event trigger function, the flow_type is implicitly PD, so both parameters flow_super_CAS_id and flow_id shall be given explicitly; in particular the flow_super_CAS_id is not always the custom_CAS_id.



NOTE: The flow PID provisioning transaction allows a CAS to receive PID values for particular streams. The CA System is responsible for the (possibly undesirable) consequences, when using these PIDs. In particular, inserting these PIDs in private data can cause problems when re-multiplexing occurs.

Figure 11: PID provision mechanism for ECM and EMM/private data streams

8.2.8 Implementation of the C(P)SIG \Leftrightarrow (P)SIG protocol

The generic C(P)SIG \Leftrightarrow (P)SIG interface defined in the clauses above shall be implemented in one of the two following ways:

- in a connection-oriented protocol: in this case the implementation shall be compliant with the clause 8.3;
- in a SIMF-based protocol: in this case, the implementation shall be compliant with the clause 8.4.

8.3 Connection-oriented protocol

This clause specifies the connection-oriented instance of the generic C(P)SIG \Leftrightarrow (P)SIG protocol defined in the clause 8.2.

In such an implementation the C(P)SIG and (P)SIG processes communicate with each other via a connection-oriented protocol.

8.3.1 Overview of the C(P)SIG \Leftrightarrow (P)SIG connection-oriented protocol

8.3.1.1 Principles

In this connection-oriented protocol, the (P)SIG is the client and the C(P)SIG is the server. The (P)SIG has a prior knowledge of the mapping between Custom_CAS_id and the IP addresses and port numbers of the C(P)SIG. Once the (P)SIG has established a TCP connection with the C(P)SIG, it opens a channel (see clause 8.3.1.2) dedicated to the Custom_CAS_id. Then it opens as many streams (see clause 8.3.1.3) as transport streams it manages. When a stream is open, the C(P)SIG may, for the relevant transport stream, request for PSI and SI tables, be triggered on particular conditions and insert private descriptors.

To achieve this, this connection-oriented protocol offers a set of messages at channel level and stream level, which can be used according to a specific state machine. Messages and state machine are described in the following clauses.

The general format of the messages is defined in clause 4.4.1. The following points also apply to all C(P)SIG \Leftrightarrow (P)SIG messages specified by this connection-oriented protocol:

- depending on their type or their use in the state machine, certain messages may be sent by a (P)SIG only (noted "C(P)SIG \Leftarrow (P)SIG"), others may be sent by a C(P)SIG only (noted "C(P)SIG \Rightarrow (P)SIG"), the remainder may be sent by either (noted "C(P)SIG \Leftrightarrow (P)SIG"). This is clearly indicated in the descriptions that follow;
- a message whose name ends with "_request" shall be answered by the receiver, using the corresponding message whose name ends with "_response";
- all messages include a 16-bit transaction_id, which is used to match messages with their responses. The transaction_id is assigned cyclically by the originator of the message (C(P)SIG or (P)SIG), and is unique per channel. Values of 0 through 32 767 are reserved for C(P)SIG use; the (P)SIG shall use values 32 768 through 65 535;
- though a real implementation will include time_out management, the present document does not specify a time-out for a response, nor the actions (e.g. failure or retry) to be taken if a message is not received within a given time. The present document also does not define any specific mechanisms to ensure message delivery other than TCP itself.

8.3.1.2 Channels

8.3.1.2.1 Definition and types

The logical connection between a C(P)SIG and a (P)SIG is comprised of one and only one channel per CAS. There is only one channel per TCP connection.

Since a channel interconnects a single C(P)SIG with a single (P)SIG, any of seven types of channels is possible; refer to the leftmost diagram in figure 12. The channel types that can actually be established depend on the respective configurations of the CAS and head-end processes. For example, a CPSISIG may be interconnected with a {PSIG+SIG} with either or both of the connections depicted in the rightmost diagram in figure 12.

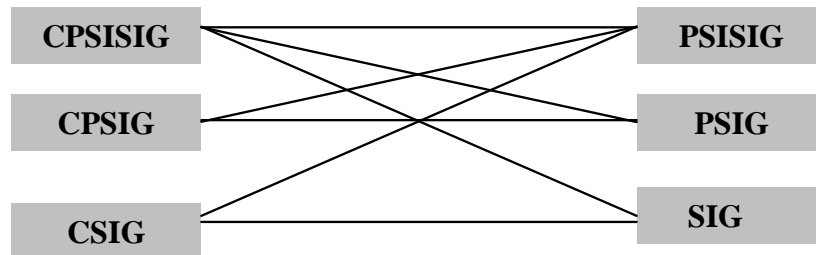


Figure 12: All types of C(P)SIG ↔ (P)SIG channels

Each channel is identified by a 2-byte custom_channel_id. This value is unique per C(P)SIG (custom_CAS_id).

8.3.1.2.2 Channel establishment

The head-end is responsible for establishing all channels at system initialization time, or when a CAS is newly interfaced to the head-end. Accordingly, the head-end (and possibly the CAS) shall have prior information of the channels to be established, as well as all parameters required for connection.

This channel information is determined by commercial agreement, per business and technical requirements. The manner in which this information is maintained and used is beyond the scope of the present document.

8.3.1.3 Streams

8.3.1.3.1 Definition

As described in clause 8.3.1.2, a channel connects a C(P)SIG with a (P)SIG. A channel logically connects a C(P)SIG with one or more TSs.

A **stream** is a logical connection between a C(P)SIG and a (P)SIG, which serves a single TS. As such, a stream "belongs" to a channel. *Within a channel*, the relationship between stream and TS is *1-to-1* (or *0-to-1*: a TS need not have any stream associated with it). Figure 13 illustrates this relationship.

The primary purpose of a stream is to serve as the logical conduit from a C(P)SIG to a (P)SIG, for the transmission of CAS-specific private data in a given TS. At least one stream shall be established, and operational, for a C(P)SIG to generate private data in any TS.

As a stream "belongs" to a channel, and a channel "belongs" to a CAS, a stream cannot convey information pertaining to more than one CAS.

Each stream is identified by a 2-byte custom_stream_id. This value is unique per channel.

The value of custom_stream_id uniquely identifies a TS. The present document is identified by the same value of custom_stream_id across all channels.

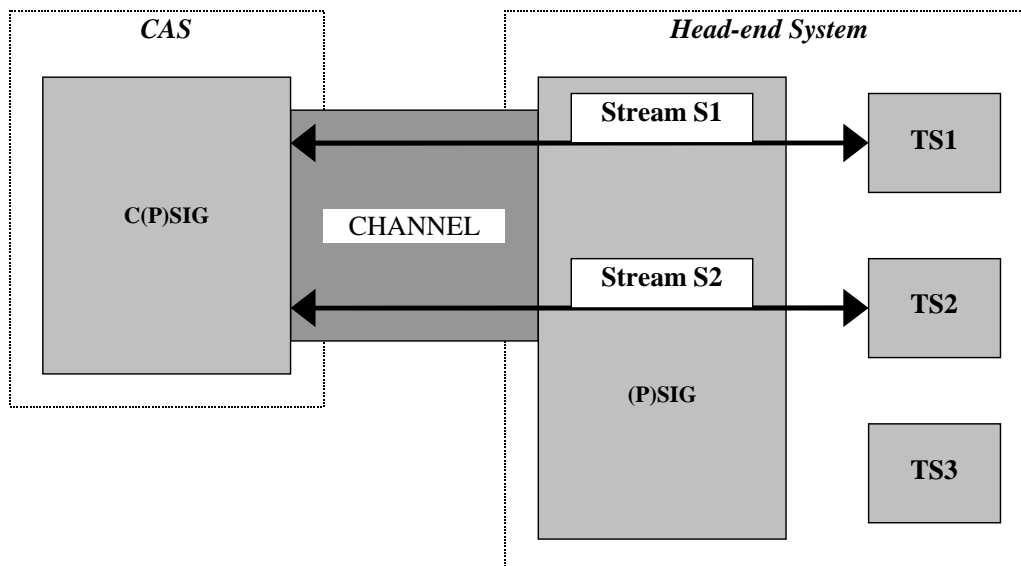


Figure 13: Stream interconnections (simple example)

Multiple streams shall be established per channel when the C(P)SIG needs to generate CAS-specific private data in more than one TS. Multiple streams corresponding to one TS cannot be configured for other reasons, e.g. performance, capacity or redundancy.

Refer also to annex F for a sample channel and stream configuration for a head-end with two CASs.

8.3.1.3.2 Stream establishment

The head-end is responsible for establishing all streams at system initialization time, or when a CAS is inserted to the head-end. Accordingly, the head-end (and possibly the CAS) shall have prior information of the streams to be established, as well as all parameters required for connection.

This stream information is determined by commercial agreement, per business and technical requirements. The manner in which this information is maintained and used is beyond the scope of the present document.

8.3.1.4 C(P)SIG \leftrightarrow (P)SIG message lists

There are two general classes of messages:

- **Channel-level messages** pertain to the configuration or status of a channel. Their semantic scope is a channel, and they do not reference streams or DVB services;

channel_setup	0x0301	C(P)SIG	\leftarrow	(P)SIG
channel_status	0x0302	C(P)SIG	\leftrightarrow	(P)SIG
channel_test	0x0303	C(P)SIG	\leftrightarrow	(P)SIG
channel_close	0x0304	C(P)SIG	\leftarrow	(P)SIG
channel_error	0x0305	C(P)SIG	\leftrightarrow	(P)SIG

- **Stream-level messages** pertain to the configuration or status of a stream, the supply of currently transmitted PSI/SI, and to the transmission of CAS-specific private data for insertion in MPEG-2/DVB tables. These messages require a channel that is established and operational. Their semantic scope is a stream, a transport stream (TS) or a DVB (or MPEG-2) service, depending on the message.

stream_setup	0x0311	C(P)SIG	⇐	(P)SIG
stream_status	0x0312	C(P)SIG	⇔	(P)SIG
stream_test	0x0313	C(P)SIG	⇔	(P)SIG
stream_close	0x0314	C(P)SIG	⇐	(P)SIG
stream_close_request	0x0315	C(P)SIG	⇒	(P)SIG
stream_close_response	0x0316	C(P)SIG	⇐	(P)SIG
stream_error	0x0317	C(P)SIG	⇔	(P)SIG
stream_service_change	0x0318	C(P)SIG	⇐	(P)SIG
stream_trigger_enable_request	0x0319	C(P)SIG	⇒	(P)SIG
stream_trigger_enable_response	0x031A	C(P)SIG	⇐	(P)SIG
trigger	0x031B	C(P)SIG	⇐	(P)SIG
table_request	0x031C	C(P)SIG	⇒	(P)SIG
table_response	0x031D	C(P)SIG	⇐	(P)SIG
descriptor_insert_request	0x031E	C(P)SIG	⇒	(P)SIG
descriptor_insert_response	0x031F	C(P)SIG	⇐	(P)SIG
PID_provision_request	0x0320	C(P)SIG	⇒	(P)SIG
PID_provision_response	0x0321	C(P)SIG	⇐	(P)SIG

All channel-level and stream-level messages shall be supported by all C(P)SIG and (P)SIG processes, with any semantic exceptions noted in the present document.

8.3.1.5 Protocol state machines definition

This protocol is based on two state machines. The first state machine defines the transitions associated with each channel. The second state machine defines the transitions associated with each stream; this state machine is valid only for a channel in the Channel Open state.

The state machines defines:

- **States:** a total of eleven states is defined, four for the channel state machine, and seven for the stream state machine;
- **Transitions:** the set of messages that cause changes of state in the state machine.

The basic functionality of each message is presented in order to understand state machine operation. Precise and detailed syntax and semantics of each message are presented in clause 8.3.2.

Independence and scalability: all channels in a complete system are mutually independent; the same is true of streams. Therefore, both state machines are scalable: concurrent state machines can be launched for each channel and stream in the system, whether the head-end is hosting one or multiple CASs.

8.3.1.6 Channel state machine

As described in clause 8.3.1.2.2, the head-end establishes and maintains one or more channels between (P)SIG and C(P)SIG processes. This clause presents the channel state machine, which defines the sequence of channel-level messages that shall be used to establish and maintain a **single channel**.

Channels may be established in any order. In addition, a (P)SIG needs not wait for the establishment of one channel to be complete, before commencing the establishment of another channel. Such considerations are out of the scope of the present document.

The channel state machine is found in figure 14. Each state found in this state machine is defined in clauses 8.3.1.6.1 to 8.3.1.6.4.

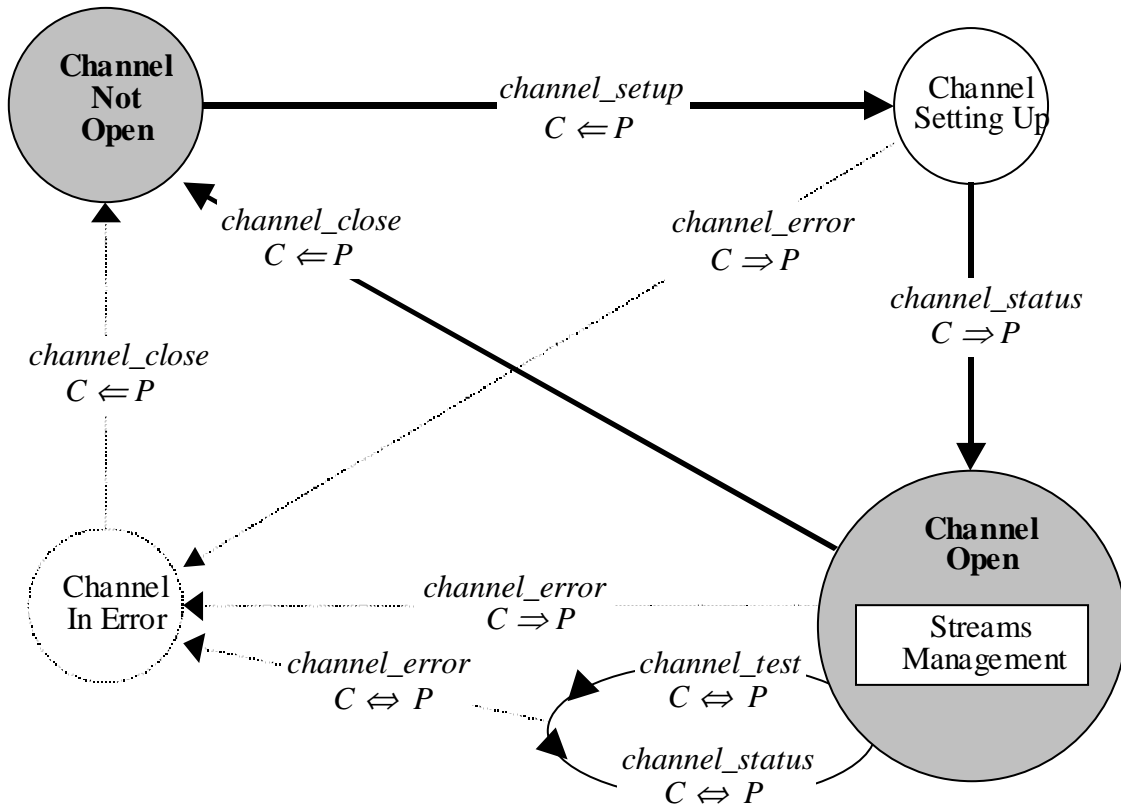


Figure 14: C(P)SIG ⇔ (P)SIG channel state machine

8.3.1.6.1 Channel Not Open

This state represents the initialization of the channel state machine. At this point, a TCP connection is assumed to be established and the channel has either not been initialized, or has been closed.

- The (P)SIG initializes a channel by sending a **channel_setup message** to the C(P)SIG on the other end of the channel. Channel_setup is the only permissible message in the Channel Not Open state. Parameters of channel_setup identify the (P)SIG type (PSISIG, PSIG or SIG) and the kinds of triggers supported by the (P)SIG for the new channel. Transmission and receipt of channel_setup move the state machine to the Channel Setting Up state.

8.3.1.6.2 Channel Setting Up

From this state, the C(P)SIG shall respond with either a channel_status or a channel_error message:

- The **channel_status** message acknowledges successful channel establishment, and that the channel is open. The C(P)SIG also indicates, via this message, the maximum number of streams that can be supported on the new channel. Transmission and receipt of channel_status move the state machine to the Channel Open state.
- The **channel_error** message acknowledges that the C(P)SIG could not open the channel, and that the channel shall be closed by the (P)SIG. One or more error codes explain the failure. Transmission and receipt of Channel_error move the state machine to the Channel In Error state.

8.3.1.6.3 Channel Open

This state represents the steady-state operation of the channel state machine. As long as the channel is open and error-free, streams may be opened, used and closed, per the stream state machine defined in clause 8.3.1.7: the stream state machine defines the stream-level and data-level messages that can be sent on a stream within the channel, per the state of that stream.

Four kinds of channel-level messages can be sent while in Channel Open state:

- either the C(P)SIG or the (P)SIG can send a **channel_test** message, in order to verify the error-free operation of the channel. This does not change the state of the channel state machine;
- if the channel is in an error-free situation, the receiver of the channel_test message shall reply with a **channel_status** message. This does not change the state of the channel state machine. Channel_status may be sent only in response to channel_test;
- if the C(P)SIG encounters an unrecoverable channel error at any other time, it shall send the (P)SIG a **channel_error** message. If the stream has unrecoverable errors, the receiver of the channel_test message shall reply with a **channel_error** message. One or more error codes explain the failure. Transmission and receipt of channel_error move the state machine to the Channel In Error state. Channel_error may be sent at any time from the Channel Open state;
- if the (P)SIG wants to close the channel for any reason, it shall send the C(P)SIG a **channel_close** message. Receipt of channel_close moves the state machine to the Channel Not Open state. Channel_close may be sent at any time from this state.

Channel_close also causes the immediate closure of all streams open in the channel.

8.3.1.6.4 Channel In Error

This temporary and short-lived state is used only to represent the fact that the C(P)SIG has encountered and reported an unrecoverable channel error. The (P)SIG shall close the channel:

- The (P)SIG sends a **channel_close** message to the C(P)SIG. Transmission and receipt of channel_close move the state machine to the Channel Not Open state.

Channel_close also causes the immediate closure of all streams open in the channel.

8.3.1.7 Stream state machine

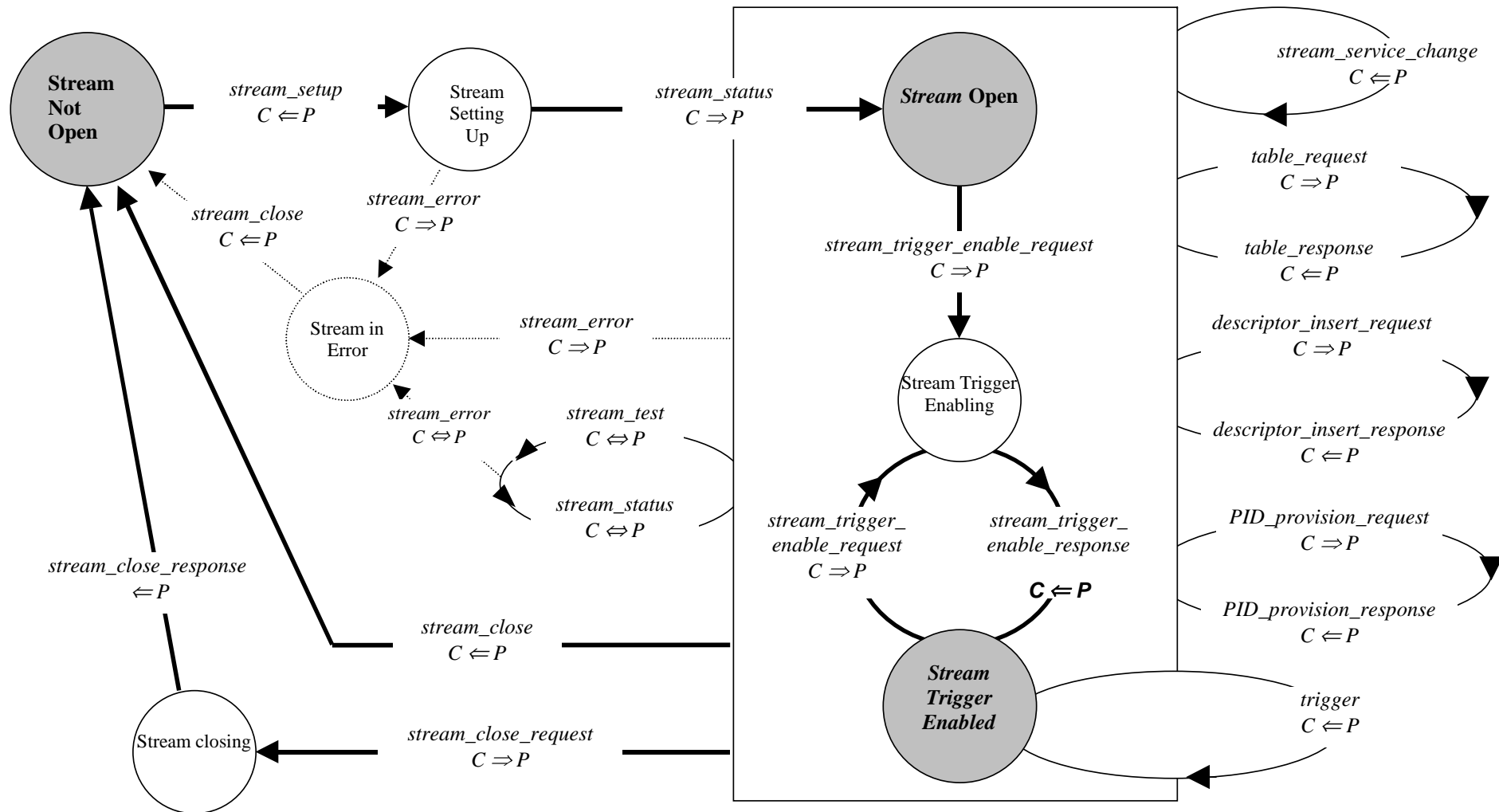
As described in clause 8.3.1.3.2, the head-end establishes one or more streams within a channel. This clause presents the stream state machine, which defines the sequence of stream-level messages that shall be used to establish, maintain and use a **single stream** within a channel.

Streams may be established in any order (within a given channel, or globally). In addition, a (P)SIG needs not wait for the establishment of one stream to be complete, before commencing the establishment of another stream. Such considerations are out of the scope of the present document.

The channel shall be in the Channel Open state (see clause 8.3.1.6.3) for a (P)SIG to initiate a stream state machine. The channel state machine, as defined in clause 8.3.1.6, continues to operate in Channel Open state during the operation of the stream state machine. Accordingly, both C(P)SIG and (P)SIG processes shall properly handle any and all channel-level messages valid in Channel Open state (these messages are not shown in the stream state machine).

Closure of a channel (Channel Not Open state) causes the immediate closure of all streams open in the channel (reset of the stream state machine to Stream Not Open state).

The stream state machine is found in figure 15. Each state found in this state machine is defined in clauses 8.3.1.7.1 to 8.3.1.7.7.



8.3.1.7.1 Stream Not Open

This state represents the initialization of the stream state machine. At this point, the stream has either not been initialized, or has been closed. The channel in which the stream is found shall be in the Channel Open state in order to proceed.

The (P)SIG initializes a stream by sending a **stream_setup message** to the C(P)SIG on the other end of the stream. **stream_setup** is the only permissible message in the Stream Not Open state. Parameters of **stream_setup** identify the transport stream (TS) associated with the stream, and list every service in the TS at the time the **stream_setup** is sent. Transmission and receipt of **stream_setup** move the state machine to the Stream Setting Up state.

8.3.1.7.2 Stream Setting Up

From this temporary and short-lived state, the C(P)SIG shall respond with either a **stream_status** or a **stream_error** message:

- the **stream_status** message acknowledges successful stream establishment, and that the stream is open. Transmission and receipt of **stream_status** move the state machine to the Stream Open state;
- the **stream_error** message acknowledges that the C(P)SIG could not open the stream, and that the stream shall be closed by the (P)SIG. One or more error codes explain the failure. Transmission and receipt of **stream_error** move the state machine to the Stream In Error state.

8.3.1.7.3 Stream Open

The stream is open and operational, and capable of performing all operations with exception of action triggering by the (P)SIG. Thirteen kinds of stream-level messages can be sent while in Stream Open state:

- the C(P)SIG sends a **stream_trigger_enable_request** message to enable the receipt of (P)SIG action triggers in the stream. Parameters of **stream_trigger_enable_request** indicate, for any or all services in the TS, which kinds of triggers the C(P)SIG wants to receive, and how long in advance of the action the C(P)SIG wants to be triggered. Transmission and receipt of **stream_trigger_enable_request** move the state machine to the Stream Trigger Enabling state;
- the C(P)SIG sends a **table_request** message to request data from any PSI, SI or private table currently transmitted. This does not change the state of the stream state machine. The (P)SIG shall respond to each **table_request** message with a **table_response** message;
- the (P)SIG sends a **table_response** message to supply data from any PSI, SI or private table currently transmitted, as requested by the C(P)SIG in a **table_request** message. This does not change the state of the stream state machine. However, *for a given stream*, the (P)SIG shall respond to **table_request** messages in the order received. **Table_response** may be sent only in response to **table_request**;
- the C(P)SIG sends a **descriptor_insert_request** message to request the insertion of CAS-specific private data in the TS. The private data is a list (possibly empty) of (a) private descriptors, and/or (b) private_data_bytes for a PAT or PMT CA_descriptor. Other parameters indicate the precise location within the TS in which to insert the data, and, optionally, with which triggered action the data insertion should be synchronized. The (P)SIG shall respond to each **descriptor_insert_request** message with a **descriptor_insert_response** message. The **descriptor_insert_request** message may be sent at any time from this state. This message does not change the state of the stream state machine;
- the (P)SIG sends a **descriptor_insert_response** message in response to the **descriptor_insert_request** message. Parameters indicate either successful data insertion in the TS or reasons for failure. The **descriptor_insert_response** message does not change the state of the stream state machine. However, *for a given stream*, the (P)SIG shall respond to **descriptor_insert_request** messages in the order received. **Descriptor_insert_response** may be sent only in response to **descriptor_insert_request**;
- the C(P)SIG sends a **PID_provision_request** message to request from the (P)SIG the PID value the head-end has assigned to a particular stream; this stream is identified by a unique identifier. The (P)SIG shall respond to each **PID_provision_request** message with a **PID_provision_response** message. The **PID_provision_request** message may be sent at any time from this state. This message does not change the state of the stream state machine;

- the (P)SIG sends a **PID_provision_response** message to provide the C(P)SIG with the PID value requested by the C(P)SIG in a **PID_provision_request** message. The **PID_provision_response** message does not change the state of the stream state machine. **PID_provision_response** may be sent only in response to **PID_provision_request**;
- the (P)SIG sends a **stream_service_change** message to signal the addition, deletion or move (TS and/or service reassignment) of a service. This message is not acknowledged by the C(P)SIG, and does not change the state of the stream state machine. Note that the head-end (any (P)SIG) shall send this message to all C(P)SIG processes operating in the Stream Open or Stream Trigger-Enabled state;
- if the (P)SIG wants to close the stream for any reason, it sends a **stream_close** message. Transmission and receipt of **stream_close** move the state machine directly to the Stream Not Open state. **stream_close** may be sent at any time from this state. This message is not acknowledged by the C(P)SIG;
- if the C(P)SIG wants to close the stream for any reason, it sends a **stream_close_request** message. Transmission and receipt of **stream_close_request** move the state machine to the Stream Closing state. **stream_close_request** may be sent at any time from this state;
- either the C(P)SIG or the (P)SIG can send a **stream_test** message, in order to verify the error-free operation of the stream. This does not change the state of the stream state machine. **stream_test** may be sent at any time from this state;
- if the stream is in an error-free situation, the receiver of the **stream_test** message shall reply with a **stream_status** message. This does not change the state of the stream state machine. **stream_status** may be sent only in response to **stream_test**;
- if the stream has unrecoverable errors, the receiver of the **stream_test** message shall reply with a **stream_error** message. One or more error codes explain the failure. Transmission and receipt of **stream_error** move the state machine to the Stream In Error state.

8.3.1.7.4 Stream Trigger Enabling

This temporary and short-lived state is entered only after the C(P)SIG sends a **stream_trigger_enable_request** message while in the Stream Open state or in Stream Trigger Enabled state. One kind of stream-level messages can be sent while in Stream Trigger Enabling state:

- The (P)SIG sends a **stream_trigger_enable_response** message in reply to the C(P)SIG. Parameters of **stream_trigger_enable_response** indicate, for any or all services in the TS, which kinds of triggers the C(P)SIG *can actually* receive, and *approximately* how long in advance of the action the C(P)SIG *will actually* be triggered. Transmission and receipt of **stream_trigger_enable_response** move the state machine to the Stream Trigger-Enabled state (whether or not any triggers are really enabled). **stream_trigger_enable_response** may be sent only in response to **stream_trigger_enable_request**.

8.3.1.7.5 Stream Trigger-Enabled

This state represents the normal steady-state operation of the stream state machine. The stream is open and operational, and capable of performing all operations, *including* (a) action triggering by the (P)SIG, and (b) requests for CAS-specific private data insertion by the C(P)SIG.

Fourteen kinds of stream-level messages can be sent while in Stream Trigger-Enabled state. The first thirteen are the same as for the Stream Open state; see clause 8.3.1.7.3. The additional type of stream-level messages is as follows:

- the (P)SIG sends a **trigger** message to signal an action of one of the following types:
 - new DVB SI EIT Following event;
 - new head-end information about a future DVB SI EIT event;
 - creation, modification or closure of an ECM stream;
 - modification of an ECM, EMM or private data PID;
 - user-defined (per commercial agreement).

Parameters indicate the trigger type, and all information required to precisely qualify the action that caused the trigger.

The trigger message is not explicitly acknowledged by the C(P)SIG. However, the trigger shall be referenced by a subsequent `descriptor_insert_request` message (see below) if the C(P)SIG wants to synchronize private data transmission with the triggering action.

The trigger message may be sent at any time from this state. This message does not change the state of the stream state machine.

8.3.1.7.6 Stream In Error

This temporary and short-lived state is used only to represent the fact that the C(P)SIG has encountered and reported an unrecoverable stream error:

- The (P)SIG sends a **stream_close** message to the C(P)SIG. Transmission and receipt of `stream_close` move the state machine to the Stream Not Open state.

8.3.1.7.7 Stream Closing

This temporary and short-lived state is used only to represent the fact that the C(P)SIG has requested closure of the stream:

- The (P)SIG sends a **stream_close_response** message to the C(P)SIG, to confirm closure of the stream. Transmission and receipt of `stream_close_response` move the state machine to the Stream Not Open state.

8.3.1.8 Summary of messages permissible in each state

Table 35 provides a listing of the channel-level and stream-level messages that may be generated in each of the states of both state machines.

Table 35: Message/state cross-reference for the C(P)SIG ↔ (P)SIG state machines

Message	State										
	Channel (assumes Stream Not Open)				Stream (assumes Channel Open)						
	Not Open	Setting Up	In Error	Open	Not Open	Setting Up	In Error	Closing	Open	Trigger Enabling	Trigger Enabled
channel_setup	X										
channel_status		X		X	X	X	X	X	X	X	X
channel_test				X	X	X	X	X	X	X	X
channel_close			X	X	X	X	X	X	X	X	X
channel_error		X		X	X	X	X	X	X	X	X
stream_setup					X						
stream_status						X			X		X
stream_test									X		X
stream_close							X		X		X
stream_close_request									X		X
stream_close_response								X			
stream_error						X			X		X
stream_trigger_enable_request									X		X
stream_trigger_enable_response										X	
stream_service_change									X		X
trigger											X
table_request									X		X
table_response									X		X
descriptor_insert_request									X		X
descriptor_insert_response									X		X
PID_provision_request									X		X
PID_provision_response									X		X

8.3.2 C(P)SIG \Leftrightarrow (P)SIG message syntax and semantics

This clause provides precise syntax and semantics for each of the messages in the C(P)SIG \Leftrightarrow (P)SIG connection-oriented protocol.

8.3.2.1 List of message parameters for the C(P)SIG \Leftrightarrow (P)SIG protocol

Table 36: Parameter syntax in C(P)SIG \Leftrightarrow (P)SIG messaged-based protocol

Parameter_type	Parameter	Type (/Units)	Length (bytes)
0x000D	access_criteria	user defined	variable
0x0100	bouquet_id	uimbsf	2
0x0101	CA_descriptor_insertion_mode	uimbsf	1
0x0102	custom_CAS_id	uimbsf	4
0x0103	custom_channel_id	uimbsf	2
0x0104	custom_stream_id	uimbsf	2
0x0105	descriptor	Per MPEG/DVB; see ISO/IEC 13818-1 [3] and EN 300 468 [1]	variable
0x0106	descriptor_insert_status	uimbsf	1
0x0107	duration	uimbsf	3
0x0108	ECM_related_data	-	variable
	ES_id		
	flow_super_CAS_id		
	flow_id		
	flow_PID		
	access_criteria		
0x0109	DVB reserved	-	-
0x010A	DVB reserved	-	-
0x010B	ES_id	uimbsf	2
0x010C	event_id	uimbsf	2
0x010D	event_related_data	-	variable
	event_id		
	duration		
	start_time		
	private_data		
0x010E	flow_id	uimbsf	2
0x010F	flow_PID	uimbsf	2
0x0110	flow_PID_change_related_data	-	9
	flow_type		
	flow_super_CAS_id		
	flow_id		
	flow_PID		
0x0111	flow_super_CAS_id	uimbsf	4
0x0112	flow_type	uimbsf	1
0x0113	insertion_delay	tcimbsf (/ms)	2
0x0114	insertion_delay_type	uimbsf	1
0x0115	last_section_indicator	boolean	1
0x0116	location_id	uimbsf	1
0x0117	max_comp_time	uimbsf (/sec)	2
0x0118	max_streams	uimbsf	2
0x0119	MPEG_section	Per MPEG/DVB; see ISO/IEC 13818-1 [3] and EN 300 468 [1]	variable
0x011A	network_id	uimbsf	2
0x011B	original_network_id	uimbsf	2
0x011C	private_data	user-defined	variable
0x011D	private_data_specifier	uimbsf	4
0x011E	(P)SIG_type	uimbsf	1
0x011F	segment_number	uimbsf	1
0x0120	service_id	uimbsf	2
0x0121	service_parameters	-	8
	service_id		
	trigger_list		
	max_comp_time		
0x0122	start_time	bslbf	5

Parameter_type	Parameter	Type (/Units)	Length (bytes)
0x0123	stream_change_timestamp	bslbf	5
0x0124	stream_change_type	uimsbf	1
0x0125	table_id	uimsbf	1
0x0126	transaction_id	uimsbf	2
0x0127	transport_stream_id	uimsbf	2
0x0128	trigger_id	uimsbf	2
0x0129	trigger_list	bslbf	4
0x012A	trigger_type	uimsbf	4
0x012B	PD_related_data	-	variable
	ES_id		
	flow_super_CAS_id		
	flow_id		
	flow_PID		
	flow_stream_type		
	private_data		
0x012C	flow_stream_type	uimsbf	1
0x012D to 0x6FFF	DVB reserved	-	-
0x7000	error_status	TBD	2
0x7001	error_information	user defined	variable
0x7002 to 0x7FFF	DVB reserved	-	-
0x8000 to 0xFFFF	user defined	-	-

8.3.2.2 Parameter semantics

This clause gives the semantic of the parameters and fields used in this protocol. When depending on the message context, some parameters can be completed in each message description in further clauses.

- **access_criteria:** this parameter carries the access criteria concerning an ECM stream;
- **bouquet_id:** this parameter is the value of the bouquet_id field as defined by EN 300 468 [1];
- **CA_descriptor_insertion_mode:** this parameter is provided by a CPSIG/CPSISIG and indicates whether a PSIG/PSISIG has to insert the skeleton of CA_descriptors; such a skeleton corresponds to the CA_descriptor as defined in table 2.51 of ISO/IEC 13818-1 [3] with an empty private_data_byte part; it can have the following values:
 - **0x01:** the skeleton of the CA_descriptor is always inserted by the PSIG/PSISIG;
 - **0x02:** no CA_descriptor skeleton is inserted by the PSIG/PSISIG; the CPSIG/CPSISIG is always responsible for the generation of this descriptor;
 - **other values:** DVB reserved.
- **custom_CAS_id:** this parameter is the unique identifier of a C(P)SIG sharing a channel with a (P)SIG;
- **custom_channel_id:** this parameter is the identifier of a channel established by a (P)SIG with a C(P)SIG. It is unique per custom_CAS_id;
- **custom_stream_id:** this parameter is the identifier of a stream established by a (P)SIG with a C(P)SIG for a transport stream. It is unique per channel;
- **descriptor:** this field represents an instance of a descriptor provided by a C(P)SIG and to be inserted by the (P)SIG;
- **descriptor_insert_status:** this parameter indicates if a descriptor insertion requested by a C(P)SIG to a (P)SIG has been done correctly or has failed, according to the following values:
 - **0x00:** insertion has been done correctly; this value is significant after the actual insertion;
 - **0x01:** insertion failed because of the request was inconsistent (e.g. bad value for location_id, lack of some parameters for a given location_id); this value is significant immediately after the descriptor insertion request and prior the actual insertion;

- **0x02:** insertion has failed because the target table is not generated by the (P)SIG; this value is significant immediately after the descriptor insertion request and prior the actual insertion;
- **0x03:** insertion has failed because an error occurred at the moment of insertion of the descriptors in the table (e.g. missing space); this value is significant after the actual insertion attempt;
- **other values:** DVB reserved.
- **duration:** this parameter contains the duration of the event in hours, minutes and seconds. The format is six 4-bit BCD digits;

EXAMPLE: 01:45:30 is coded as 0x014530.

- **ECM_related_data:** this field provides the C(P)SIG with the necessary information concerning an ECM stream that is going to be opened, closed or modified, whose PID is about to change, or that is attached to an event. It includes the following subfields:
 - ES_id;
 - flow_id: the identifier of the ECM stream;
 - flow_super_CAS_id: the Super_CAS_id the ECM belongs to;
 - flow_PID: the PID value of the concerned ECM stream;
 - access_criteria.
- **error_information:** this parameter describes an error reason; its values are given in clause 8.3.4.16;
- **error_status:** this parameter describes an error reason; its values are given in clause 8.3.4.16;
- **ES_id:** this parameter is the identifier either of an Elementary Stream to which the ECM stream is attached or of the PD stream in the PMT. It is equal to the rank of the description of the Elementary Stream, respectively of the PD stream, in the PMT. The value 0 identifies the whole service;
- **event_id:** this parameter is the value of the event_id field as defined by EN 300 468 [1];
- **event_related_data:** this field describes a DVB-event. The trigger message does not contain this parameter when the value of trigger type is different from "following event" and "future event". It comprises the following subfields:
 - event_id;
 - duration;
 - start_time;
 - private_data.
- **flow_id:** this parameter uniquely identifies a stream for a given stream type and a given flow_super_CAS_id;
- **flow_PID:** this parameter is the PID value of a stream. It is left-padded with zeroes to fill 2 bytes;
- **flow_PID_change_related_data:** this field provides the C(P)SIG with the necessary information concerning an ECM, EMM or private data stream whose PID is about to change. It includes the following subfields:
 - flow_type;
 - flow_super_CAS_id;
 - flow_id;
 - flow_PID.
- **flow_super_CAS_id:** this parameter identifies the CA system and the CA subsystem a stream belongs to;

- **flow_type:** this parameter identifies the type of a stream:
 - **0x00:** EMM;
 - **0x01:** private data;
 - **0x02:** ECM;
 - **other values:** DVB reserved.
- **flow_stream_type:** Stream type value of the PD stream as described in the PMT;
- **insertion_delay:** this parameter gives the amount of time between the time of a trigger cause and the time at which the descriptors associated to this trigger cause should be inserted. If insertion_delay is negative, the descriptor insertion shall be done before the trigger cause;
- **insertion_delay_type:** this parameter indicates to the (P)SIG timing requirements for transmission of the updated table, with respect to the event start time or ECM modification (as determined by the trigger_type parameter). This parameter can have either of these values:
 - **0x01** = "immediate": the (P)SIG shall immediately insert the set of descriptors in the table;
 - **0x02** = "synchronized": the (P)SIG shall synchronize the insertion of the set of descriptors with the cause identified by trigger_id. An insertion_delay parameter indicates the amount of time between this cause and the time of the insertion of the descriptors in the table;
 - **other values:** DVB reserved.
- **last_section_indicator:** this parameter is a boolean used in a set of messages carrying table sections to C(P)SIG; when true in a message, it indicates that this message is the last one in the sequence of responses;
- **location_id:** this parameter identifies the table and descriptor loop (if applicable) in which a (P)SIG is to insert a set of descriptors. See table 32 for values;
- **max_comp_time:** this parameter defines the delay for which a trigger precedes the corresponding scheduled action; depending on message context this delay is the one estimated by a (P)SIG or the one wanted by a C(P)SIG;
- **max_streams:** this parameter gives the maximum number of streams supported for a channel by a C(P)SIG or a (P)SIG, depending on message context;
- **MPEG_section:** this parameter includes one and only one MPEG-2 PSI or DVB-SI section;
- **network_id:** this parameter is the value of the network_id field as defined by EN 300 468 [1];
- **original_network_id:** this parameter is the value of the original_network_id field as defined by EN 300 468 [1];
- **private_data:** this parameter corresponds to the event related private data. Its contents are private;
- **private_data_specifier:** this parameter contains the value of a private_data_specifier as defined by EN 300 468 [1] and ETR 162 [i.2];
- **PD_related_data:** this field provides the C(P)SIG with the necessary information concerning a Private Data stream that is going to be opened, closed or modified. It includes the following subfields:
 - ES_id;
 - flow_id: the identifier of the PD stream;
 - flow_super_CAS_id: the Super_CAS_id the PD belongs to;
 - flow_PID: the PID value of the concerned PD stream;

- flow_stream_type;
- possible private data.
- **(P)SIG_type:** depending on the message context this parameter identifies a (P)SIG as a PSISIG, PSIG or SIG or a C(P)SIG as a CPSISIG, CPSIG or CSIG. See table 29 for values;
- **segment_number:** this field represents the number of the segment of an EIT schedule. Refer to TR 101 211 [i.3] for the definition of a segment;
- **service_id:** this parameter is the value of the service_id field as defined by EN 300 468 [1] for a DVB service, or the PAT program_number of an MPEG-2 program; this parameter allows to refer to an MPEG2 program, which is not a DVB service;
- **service_parameters:** this field includes a set of the following subfields:
 - service_id;
 - trigger_list;
 - max_comp_time.
- **start_time:** this parameter contains the start time of the event in Modified Julian Date (MJD) and Universal Time, Coordinated (UTC) formats. This 5-byte field is coded as 40 bits: the 16 LSBs of MJD, followed by six 4-bit Binary Coded Decimal (BCD) digits representing UTC. Example: 93/10/13 12:45:00 is coded as 0xC079124500;
- **stream_change_timestamp:** this field signals the scheduled date and time of a stream change. The format is the same as the one of start-time field;
- **stream_change_type:** this parameter signals the type of stream change being made:
 - **0x01:** service creation;
 - **0x02:** service deletion;
 - **other values:** DVB reserved.
- **table_id:** this parameter is the value of the table_id field as defined by ISO/IEC 13818-1 [3];
- **transaction_id:** this parameter is the unique identifier of a command; depending on message context, this parameter may allow to identify a corresponding response;
- **transport_stream_id:** this parameter is the value of the transport_stream_id as defined by ISO/IEC 13818-1 [3];
- **trigger_id:** this parameter uniquely identifies a trigger occurrence. Such a trigger_id defined in a message may be referred to later on by another message;
- **trigger_list:** this parameter is a 32-bit vector, each bit of which corresponds to a trigger type, according to table 30. A bit set to 1 means that the corresponding trigger type is concerned. Depending on message context this vector gives:
 - either the intrinsic trigger possibilities of a (P)SIG (at channel setup);
 - or the trigger types a C(P)SIG wants to receive (at trigger enable request);
 - or the effective trigger types a (P)SIG can generate according to its intrinsic possibilities and to the C(P)SIG requests.
- **trigger_type:** this parameter identifies the type of trigger. See table 30 for values.

8.3.3 Channel-level messages

This clause defines the syntax and semantics of each channel-level message. It refers to the syntax and semantic of parameters given in clause 8.3.2; however for some parameters according to a particular message context, complementary descriptions can be given to complete or to replace descriptions given in clause 8.3.2.

8.3.3.1 Channel_setup message: C(P)SIG \Leftarrow (P)SIG

The channel_setup message is sent by a (P)SIG from the Channel Not Open state, to establish a channel with a C(P)SIG.

Parameter	Number of instances in message
transaction_id	1
custom_CAS_id	1
custom_channel_id	1
(P)SIG_type	1
trigger_list	1
max_streams	1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

(P)SIG_type: identifies the (P)SIG as a PSIG, SIG or PSISIG.

trigger_list: gives the trigger types the (P)SIG can *actually* generate.

max_streams: informs the C(P)SIG as to the maximum number of streams supported by the (P)SIG for this channel.

8.3.3.2 Channel_status message: C(P)SIG \Leftrightarrow (P)SIG

The channel_status message is used either:

- by a C(P)SIG from the Channel Setting Up state, to indicate successful channel setup;
- by either a C(P)SIG or a (P)SIG from the Channel Open state, to indicate the status of a channel. This message follows a channel_test message issued by the process that shares the channel in question when no error has been detected.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
(P)SIG_type	1
trigger_list	1
max_streams	1
CA_descriptor_insertion_mode	0 to 1

transaction_id: is set to the transaction_id of the message to which channel_status refers. This is the transaction_id of the previous channel_setup or a channel_test message.

(P)SIG_type: identifies the type of the message sender as a PSISIG, PSIG or SIG if (P)SIG or as a CPSISIG, CPSIG or CSIG if C(P)SIG.

trigger_list: lists the kinds of action triggers supported by the (P)SIG for this channel. When channel_status is issued by the C(P)SIG (as a result of a channel_setup or a channel_test issued by the (P)SIG), this parameter shall be ignored by the (P)SIG.

max_streams: is used in one of two ways:

- when channel_status is a response to channel_setup, the C(P)SIG imposes its max_streams limitation on the (P)SIG. This shall be less than or equal to the value of max_streams supplied by the (P)SIG during channel_setup;
- when channel_status is a response to channel_test, max_streams defines the maximum number of streams supported on this channel. This is the value of max_streams returned by the C(P)SIG after channel_setup.

CA_descriptor_insertion_mode: this parameter is provided by the CPSIG/CPSISIG and indicates whether the PSIG/PSISIG has to insert the skeleton of the CA_descriptor; this skeleton corresponds to the CA_descriptor as defined in table 2.51 of ISO/IEC 13818-1 [3] with an empty private_data_byte part; this parameter is mandatory and significant only when channel_status is issued by the CPSIG as response to a channel_setup command, otherwise it shall be ignored or can be omitted; it can have the following values:

- **0x01:** the skeleton of the CA_descriptor is always inserted by the PSIG;
- **0x02:** no CA_descriptor skeleton is inserted by the PSIG; the CPSIG is always responsible for the generation of this descriptor;
- **other values:** DVB reserved.

8.3.3.3 Channel_test message: C(P)SIG \Leftrightarrow (P)SIG

The channel_test message is sent by either a C(P)SIG or a (P)SIG from the channel Open state, to verify that:

- the TCP connection is still alive;
- the channel is in an error-free condition.

The receiver of the channel_test message shall reply with a channel_status message if the channel is free of errors, or a channel_error message if errors occurred.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

8.3.3.4 Channel_close message: C(P)SIG \Leftarrow (P)SIG

The channel_close message is sent by the (P)SIG to indicate that the channel is to be closed. This message is not acknowledged.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1

transaction_id: is the unique identifier of an instance of this command.

8.3.3.5 Channel_error message: C(P)SIG \Leftrightarrow (P)SIG

The channel_error message is sent by the recipient of a channel_test message or by the C(P)SIG at any time to indicate that an unrecoverable channel error occurred.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
error_status	1 to n
error_information	0 to n

If this message is generated by the C(P)SIG on unrecoverable error, **transaction_id** is the unique identifier of an instance of this command; if this message is a response to a previous channel_test message, **transaction_id** is set to the transaction_id of this channel_test message.

8.3.4 Stream-level messages

This clause defines the syntax and semantics of each stream-level message. It refers to the syntax and semantic of parameters given in clause 8.3.2; however for some parameters according to a particular message context, complementary descriptions can be given to complete descriptions given in that clause.

8.3.4.1 stream_setup message: C(P)SIG \leftarrow (P)SIG

The stream_setup message is sent by a (P)SIG from the Stream Not Open state, to establish a new stream with a C(P)SIG. This message carries the values of service_id of all the services broadcast in that transport stream.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
network_id	1
original_network_id	1
transport_stream_id	1
service_id	0 to n

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

The transport stream associated with this **custom_stream_id** is identified by **network_id**, **original_network_id** and **transport_stream_id**.

Each service (DVB service or MPEG2 program, see service_id parameter description in clause 8.3.2.2) present in the present document shall be described by a **service_id** parameter. The (P)SIG is required to signal NVOD reference services in this manner, even if it does not support private data insertion synchronized with NVOD reference events. The stream_service_change message is used to signal the addition or deletion of a service.

8.3.4.2 Stream_status message: C(P)SIG \Leftrightarrow (P)SIG

The stream_status message is used either:

- by a C(P)SIG from the Stream Setting Up state, to indicate successful stream setup;
- by either a C(P)SIG or a (P)SIG from the Stream Open, Stream Trigger Enabling, or Stream Trigger-Enabled state, to indicate the status of a stream. This message follows a stream_test message issued by the process that shares the stream in question.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
service_id	0 to n

transaction_id: is set to the transaction_id of the message to which stream_status refers. This is the transaction_id of the previous stream_setup or a stream_test message.

All **service_id** parameters concerning the present document shall be supplied, in a same way as for the stream_setup message, when stream_status is issued by a (P)SIG. When stream_status is issued by the C(P)SIG (as a result of a stream_setup or a stream_test issued by the (P)SIG), this parameter is not used.

8.3.4.3 Stream_test message: C(P)SIG \Leftrightarrow (P)SIG

The stream_test message is sent by either a C(P)SIG or a (P)SIG from the Stream Open state, to verify that the stream is in an error-free condition.

The receiver of the stream_test message shall reply with a stream_status message if the stream is free of errors, or a stream_error message if unrecoverable errors have occurred.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

8.3.4.4 Stream_close message: C(P)SIG \Leftarrow (P)SIG

The stream_close message is sent by the (P)SIG to indicate that the stream is to be closed. This message is not acknowledged by the C(P)SIG.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1

transaction_id: is the unique identifier of an instance of this command.

8.3.4.5 Stream_close_request message: C(P)SIG \Rightarrow (P)SIG

The stream_close_request message is sent by the C(P)SIG to request the (P)SIG to close a stream.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

8.3.4.6 Stream_close_response message: C(P)SIG \Leftarrow (P)SIG

The stream_close_response message is sent by the (P)SIG in response to a stream_close_request message.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1

transaction_id: is set to the transaction_id of the message to which stream_close_response refers. This is the transaction_id of the previous stream_close_request message.

8.3.4.7 Stream_error message: C(P)SIG \Leftrightarrow (P)SIG

The stream_error message is sent by the recipient of a stream_test message or by the C(P)SIG at any time to indicate that an unrecoverable stream error had occurred.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
error_status	1 to n
error_information	0 to n

If this message is generated by the C(P)SIG on unrecoverable error, **transaction_id** is the unique identifier of an instance of this command; if this message is a response to a previous stream_test message, **transaction_id** is set to the transaction_id of this stream_test message.

8.3.4.8 Stream_service_change message: C(P)SIG \Leftarrow (P)SIG

The stream_service_change message is sent by the (P)SIG to signal a modification in service existence in a TS (addition of a service to a TS or deletion of a service from a TS).

More sophisticated functions such as service_id change or the move of a service from one TS to another TS can be achieved by combining these two basic addition and deletion functions.

The head-end (any (P)SIG) shall send a stream_service_change message to all C(P)SIG processes operating in the Stream Open or Stream Trigger-Enabled state.

This message should be issued by the (P)SIG no later than the scheduled service change. The advance notification time for this message is not defined by the present document.

This message is not acknowledged by the C(P)SIG.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
service_id	1
stream_change_type	1
stream_change_timestamp	0 or 1

transaction_id: is the unique identifier of an instance of this message.

8.3.4.9 Stream_trigger_enable_request message: C(P)SIG \Rightarrow (P)SIG

The C(P)SIG sends this message to the (P)SIG for one of these reasons:

- Notify the (P)SIG that it is ready to receive action triggers, which ones it wants to receive, and (if possible) how long before the scheduled action it would like to receive the trigger. This happens after stream setup, when the stream state machine is in the stream_open state. Transmission of this message, and its receipt by the (P)SIG, move the stream state machine to the stream_trigger_enabling state. The (P)SIG is required to response with a stream_trigger_enable_response message, whose transmission and receipt move the stream state machine to the stream_trigger-enabled state.
- Change the information supplied to the (P)SIG in the most recent stream_trigger_enable_request message. This happens when the stream state machine is in the stream_trigger_enabled state. Transmission of this message and its receipt by the (P)SIG move the stream state machine to the stream_trigger_enabling state. The (P)SIG is required to response with a stream_trigger_enable_response message, whose transmission and receipt move the stream state machine to the stream_trigger-enabled state back.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
service_parameters	0 to n

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

The C(P)SIG may define for any service in the TS the trigger conditions it wants. For such each service, the C(P)SIG has to supply a **service_parameters** field including:

- a **service_id** parameter to identify the service;
- a **trigger_list** parameter to identify by which causes the C(P)SIG wants to be triggered;
- a **max_comp_time** parameter to define how far in advance of a scheduled action the C(P)SIG wants to receive the corresponding trigger.

8.3.4.10 Stream_trigger_enable_response message: C(P)SIG \Leftarrow (P)SIG

The (P)SIG shall send this message in reply to a stream_trigger_enable_request message from the C(P)SIG. This message describes the types action triggers it can actually send to the C(P)SIG, on a per-service basis.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
service_parameters	0 to n

transaction_id: is set to the transaction_id of the stream_trigger_enable_request message to which this message refers.

For each service specified by the C(P)SIG in the preceding stream_trigger_enable_request message, the (P)SIG has to supply a **service_parameters** field including:

- a **service_id** parameter to identify the service;
- a **trigger_list** parameter to identify for which causes the (P)SIG can generate triggers. This trigger_list is the logical AND of the following:
 - the trigger_list from the preceding stream_trigger_enable_request message, which represents the triggers that the C(P)SIG would like to receive;
 - a similar vector of the (P)SIG's capabilities for the current channel. This latter vector is supplied as the trigger_list in the channel_setup message;
 - a logical vector where "0" bits represent the impossibility of generating certain trigger types for certain kinds of services.
- a **max_comp_time** parameter to define how far in advance of a scheduled action the (P)SIG might possibly, given sufficient advance notification, send the corresponding trigger. The actual advance notification is estimated by the (P)SIG; the present document does not provide any mechanisms to guarantee the accuracy of this value.

8.3.4.11 Trigger message: C(P)SIG \Leftarrow (P)SIG

The trigger message is sent by the (P)SIG to the C(P)SIGs under any of the circumstances, identified by the value for the trigger_type parameter: it is generated according to the trigger types allowed in the relevant stream_trigger_enable_response message.

This message is not acknowledged.

The content of the trigger message depends on the trigger_type value, as shown in the following table.

Parameter	Number of instances in message Following event	Number of instances in message Future event	Number of instances in message ECM related event (all cases)	Number of instances in message PID change	Number of instances in message PD related event
transaction_id	1	1	1	1	1
custom_channel_id	1	1	1	1	1
custom_stream_id	1	1	1	1	1
service_id	1	1	1	0 or 1	1
trigger_id	1	1	1	1	1
trigger_type	1	1	1	1	1
event_related_data	1	1	0	0	0
ECM_related_data	0 to n	0	1	0	0
flow_PID_change_related_data	0	0	0	1	0
PD_related_data	0	0	0	0	1

When the trigger message pertains to following event, it shall contain as many ECM_related_data fields as there are ECM stream references (i.e. CA-descriptors) pertaining to the relevant CAS in the whole service description (i.e. in PMT).

In a PID change trigger for an EMM stream, the service-id parameter shall be omitted; for other stream types, this parameter shall be included.

transaction_id: is the unique identifier of an instance of this message.

trigger_id: uniquely identifies this trigger occurrence: this trigger_id may be referred to later on by the descriptor_insert_request message.

8.3.4.12 Table_request message: C(P)SIG ⇒ (P)SIG

The table_request message is sent by the C(P)SIG to request MPEG-2 PSI or DVB SI table data from the (P)SIG. The value of the custom_stream_id parameter identifies the transport_stream on which the table is broadcast.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
table_id	1
network_id	0 or 1
transport_stream_id	0 or 1
original_network_id	0 or 1
service_id	0 or 1
bouquet_id	0 or 1
event_id	0 or 1
segment_number	0 or 1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

table_id: is the value of the table_id field as defined by ISO/IEC 13818-1 [3]. It identifies the sub-table requested by the CPSIG.

The C(P)SIG requests either complete PSI tables or SI sub-tables (see EN 300 468 [1]). As EIT Schedule sub-tables may be very large, the protocol optionally allows the C(P)SIG to request only the part of the EIT Schedule sub-table that concerns a specific event or a specific segment. In case the request concerns a specific event_id, the value of the table_id field shall be arbitrarily set to 0x50 for the events pertaining to the current TS, and 0x60 for the events pertaining to an EIT other.

Table 31 indicates which parameters (**network_id**, **original_network_id**, **transport_stream_id**, **service_id**, **bouquet_id**, **event_id**, **segment_number**) shall be present in the message, depending on the type of PSI/SI table data requested by the C(P)SIG. According to this table, the transport_stream_id and original_network_id parameters identifying the transport stream carrying the requested table are implicitly included in the custom_stream_id parameter.

8.3.4.13 Table_response message: C(P)SIG \Leftarrow (P)SIG

The table_response message is sent by the (P)SIG as a response to the table_request message.

If the table does not fit into a single table_response message, the (P)SIG shall split the requested sub-table into several table_response messages with the same **transaction_id**. In every case the last table_response message for a table is indicated with the **last_section_indicator** set to "true".

When this message is a response to a request for EIT data pertaining to a single event, the sections that are returned with this message are limited to the sections containing data pertaining to that event. Likewise, when this message is a response to a request for EIT data pertaining to a single segment, the sections that are returned with this message are limited to the sections containing data pertaining to that segment, i.e. 8 sections are returned.

A single instance of a table_response may carry several **MPEG_sections**.

If there is no **MPEG_section**, this means that the table that is requested is not generated by the (P)SIG.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
last_section_indicator	1
MPEG_section	0 to n

transaction_id: is set to the transaction_id of the table_request to which this message refers.

8.3.4.14 Descriptor_insert_request message: C(P)SIG \Rightarrow (P)SIG

The descriptor_insert_request message is sent by the C(P)SIG. It indicates to the (P)SIG the descriptors that the C(P)SIG wants to be inserted in the PSI/SI tables.

The descriptors in this message cancel and replace all the existing descriptors in the same location, uniquely associated to the same custom_CAS_id and for the same private_data_specifier if given in the message. In particular, using an empty list removes all descriptors of the custom_CAS_id in that location.

This message carries all the information that the (P)SIG needs to know where and when to insert the descriptors. The table below describes the parameters that need to be present in the message, depending on the PSI/SI table which is addressed.

Depending on the needs of the C(P)SIG, this message can be sent asynchronously or as a consequence of a previous trigger message. In the latter case, it contains the trigger_id parameter.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
trigger_id	0 or 1
insertion_delay_type	1
insertion_delay	0 or 1
location_id	1
bouquet_id	0 or 1 (see table 32)
network_id	0 or 1 (see table 32)
original_network_id	0 or 1 (see table 32)
transport_stream_id	0 or 1 (see table 32)
service_id	0 or 1 (see table 32)
event_id	0 or 1 (see table 32)
ES_id	0 or 1 (see table 32)
private_data_specifier	0 or 1
descriptor	0 to n

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

custom_stream_id: identifies the transport stream on which the table that has to be updated, is broadcast.

trigger_id: refers to a previous trigger message. This field is optional. When not in the message the request for insertion is asynchronous.

The value of **insertion_delay_type** can be "synchronized" only when the descriptor_insert_request message is related to a trigger message, via the trigger_id value.

private_data_specifier: this parameter contains the value of a private_data_specifier (see EN 300 468 [1] and ETR 162 [i.2]); when this optional parameter is supplied, one of two cases applies:

- if the (P)SIG is able to generate a private_data_specifier_descriptor, the (P)SIG shall insert the following set of descriptors in the scope of a private_data_specifier_descriptor it generates with the value given by the private_data_specifier parameter;
- if the (P)SIG cannot generate a private_data_specifier_descriptor, the private_data_specifier parameter is ignored.

Table 32 indicates, for each MPEG-2 PSI and DVB SI table, which parameters (**location_id**, **bouquet_id**, **network_id**, **original_network_id**, **transport_stream_id**, **service_id**, **event_id**, **ES_id**) are needed to define where the descriptor given in the **descriptor** parameter shall be inserted. According to this table, the transport_stream_id and original_network_id parameters identifying the transport stream carrying the targeted table are implicitly included in the custom_stream_id parameter.

8.3.4.15 Descriptor_insert_response message: C(P)SIG \leftarrow (P)SIG

The (P)SIG shall send this message in reply to a descriptor_insert_request message from the C(P)SIG. This message describes the status of the descriptor insertion.

This message can be sent immediately after the reception of the corresponding request, or after the actual insertion has been attempted or completed.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
descriptor_insert_status	1

transaction_id: is set to the transaction_id of the descriptor_insert_request message to which this message refers.

8.3.4.16 PID_provision_request message: C(P)SIG \Rightarrow (P)SIG

The PID_provision_request message is sent by the C(P)SIG to receive from the (P)SIG the current PID value of a stream identified by the combination {flow_type, flow_super_CAS_id, flow_id}.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
flow_type	1
flow_super_CAS_id	1
flow_id	1

transaction_id: is the unique identifier of an instance of this command, allowing to identify the corresponding response.

8.3.4.17 PID_provision_response message: C(P)SIG \Leftarrow (P)SIG

The (P)SIG shall send this message in reply to a PID_provision_request message from the C(P)SIG. This message gives the current PID value of a flow identified by the combination {flow_type, flow_super_CAS_id, flow_id}.

Parameter	Number of instances in message
transaction_id	1
custom_channel_id	1
custom_stream_id	1
flow_type	1
flow_super_CAS_id	1
flow_id	1
flow_PID	1

transaction_id: is set to the transaction_id of the PID_provision_request message to which this message refers.

8.3.5 Error status and error information

NOTE: TCP connection level errors are beyond the scope of the present document. Only channel, stream and application level errors are dealt with. These errors occur during the lifetime of a TCP connection.

Table 37: C(P)SIG connection-oriented protocol error values

error_status value	Error type
0x0000	DVB Reserved
0x0001	invalid message
0x0002	unsupported protocol version
0x0003	unknown message_type value
0x0004	message too long
0x0005	unknown custom_stream_id value
0x0006	unknown custom_channel_id value
0x0007	too many channels on this C(P)SIG
0x0008	too many data streams on this channel
0x0009	too many data streams on this C(P)SIG
0x000A	unknown parameter_type
0x000B	unknown transaction_id value in response message
0x000C	not compliant C(P)SIG- and (P)SIG-types
0x000D	invalid value for DVB parameter
0x000E	unknown custom_CAS_id value
0x000F	unknown bouquet_id value
0x0010	invalid CA_descriptor_insertion_mode value
0x0011	invalid descriptor_insert_status
0x0012	inconsistent ES_id value
0x0013	unknown event_id value
0x0014	unknown flow_id value
0x0015	unknown flow_super_CAS_id
0x0016	invalid flow_type
0x0017	invalid insertion_delay_type value
0x0018	invalid location_id value
0x0019	unknown network_id value
0x001A	unknown original_network_id value
0x001B	invalid (P)SIG or C(P)SIG value
0x001C	invalid stream_change_type value
0x001D	invalid table_id value
0x001E	unknown transport_stream_id value
0x001F	invalid trigger_type value
0x0020	unknown service_id
0x0021	missing_service_id
0x0022	inconsistent length for DVB parameter
0x0023	missing mandatory DVB parameter
0x0024	invalid PID value
0x0025	unexpected trigger message
0x0026	unknown segment_number
0x0027	unknown trigger_id value
0x0028	unknown private_data_specifier value
0x0029	unknown descriptor_insert_status value
0x002A	custom_channel_id value already in use
0x002B	custom_stream_id value already in use
0x002C	flow_id value already in use
0x002D to 0x6FFF	DVB Reserved
0x7000	unknown error
0x7001	unrecoverable error
0x7002 to 0x7FFF	DVB Reserved
0x8000 to 0xFFFF	head-end specific/CA system specific/User defined

8.4 SIMF-based protocol

This clause specifies the SIMF-based implementation of the C(P)SIG \Leftrightarrow (P)SIG interface. This implementation is based on the (P)SI Object Information Group of the Simulcrypt Identification Module (SIM) and the Operations Reference Point (ORP) operations paradigm that can be used between any components within Simulcrypt (see clause 7):

- this operations paradigm is introduced in clause 8.4.1;
- its application to the C(P)SIG \Leftrightarrow (P)SIG interface and the conceptualization of the (P)SI object information group are introduced in clause 8.4.2;
- the (P)SI object information group is defined in clause 8.4.2.4.

8.4.1 Operations Reference Points (ORPs)

The information exchange and synchronization between any number components is accomplished through the SIMF as follows:

- the component advertises information such as SI/PSI information through a Management Information Base (examples SNMPv2 SMI MIB); that is done through a software agent (e.g. SNMPv2 agent) or through an object server (e.g. CORBA object server or RMI object server) or a web server;
- the component implements the SIMF events and logs modules which enables the asynchronous notification capability of the management system;
- the CAS Manager reads the Head-end/Uplink MIBs on a periodic basis, updates its custom information and registers itself as a recipient of events caused by updates in the Head-end/Uplink MIB;
- the CAS Manager updates the Head-end/Uplink with CA information on a periodic basis or upon receiving an event from the Head-end/Uplink or another CAS component through one of the following means:
 - sets the Uplink MIBs through the network management protocol;
 - updates its own MIBs, which causes events that result in notifications to be sent to the Head-end/Uplink manager.

Figure 16 illustrates this synchronization and information exchange mechanism within the Uplink/Head-end.

An Operation Reference Point (ORP) defines the interface between two Head-end/Uplink components. It replaces a custom communications protocol, which would need to be designed for that particular interface by a generic information exchange and event notification mechanism.

The four basic ingredients, which enable ORP, based information exchange and synchronization are:

- information access through a standardized management protocol such as SNMP or CMIP or through a standardized OO protocol such as CORBATM IIOP or JavaTM RMI;
- information advertisement through a management information base or an object broker;
- event trigger configuration, event forwarding configuration, and event notifications;
- event logging.

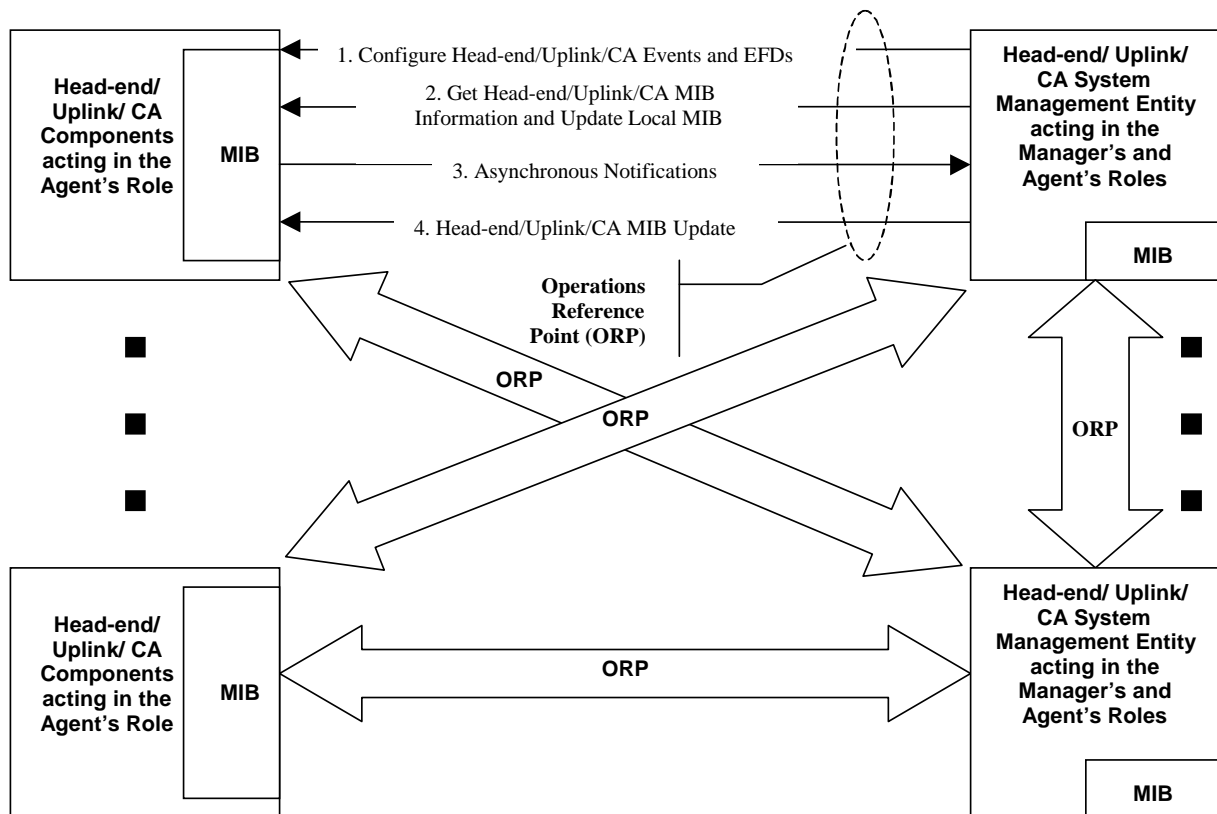


Figure 16: Operations Reference Points

8.4.2 Application of ORPs to the C(P)SIG \Leftrightarrow (P)SIG Interface

Information exchange and synchronization between C(P)SI and (P)SI generators is accomplished as follows using ORPs:

- the (P)SIG implements three SNMPv2 SMI modules:
 - events module as specified in Common Information Modules Section;
 - logs module as specified in Common Information Modules Section;
 - (P)SI/C(P)SI information within the (P)SIG which is defined in a group consisting of 9 tables as follows:
 - (P)SIG Information: this table is used to identify the different (P)SIG communications profiles; there is one entry in this table for each transport stream supported by the (P)SIG; if different (P)SIG configurations are supported for the same transport stream, there is one entry in this table for each of these configurations;
 - (P)SIG Configuration: this table is used by the C(P)SIG to configure the (P)SIG with information defining triggering parameters for ECM, EMM, and Event triggers;
 - ECM Trigger Table: this table contains all currently active ECM triggers;
 - Flow PID Change Trigger Table: this table contains all currently active Flow PID change triggers;
 - Event Trigger Table: this table contains all currently active event triggers;
 - Descriptor Insert Table: this is the table used by the C(P)SIG to communicate descriptor insertion information to the (P)SIG;
 - Descriptor Insert Descriptor Table: this is a sub-table of the Descriptor Insert Table containing the actual descriptors to be inserted;

- Table Provisioning Request Table: this is the table used by the C(P)SIG to request table provisioning by the (P)SIG;
- PID Provisioning Table: this is the table used by the C(P)SIG to request Flow PID provisioning by the (P)SIG.

The five transaction types across the C(P)SIG \leftrightarrow (P)SIG ORP are:

- 1) ECM/Event/Flow Change Triggering: An event occurs within the (P)SIG which causes asynchronous information (e.g. (P)SI tables) to be communicated to the C(P)SIG and which may be followed up by the C(P)SIG inserting descriptors into the (P)SIG's (P)SI information;
- 2) (P)SI Table Provisioning: The C(P)SIG requests and obtains (P)SI information from the (P)SIG;
- 3) (P)SI Descriptor Insertion: The C(P)SIG requests that a descriptor be inserted into the (P)SIG's (P)SI tables;
- 4) Transport Stream Service Changes: A transport stream service change is signalled by the (P)SIG by changing a variable in its static configuration table. All interested C(P)SIGs can receive value change notifications by configuring the (P)SIGs EFD table;
- 5) PID Provisioning: The C(P)SIG requests and gets the current PID value of a particular stream identified by its unique identifier.

8.4.2.1 ECM/Event/Flow Change Triggering

The (P)SI/C(P)SI ORP facilitates ECM/Event/Flow Change Triggering as follows:

- 1) the (P)SIG's Event Table is configured from start-up to generate events whenever there is a new entry in any of the three Trigger tables (i.e. the ECM, and Event Trigger tables); for each type of Trigger there is a different type of event;
- 2) the (P)SIG advertises which types of triggers it supports through a SIM variable;
- 3) the C(P)SIG configures the (P)SIG's Configuration Table with triggering parameters such as the types of triggers it wishes to receive, delay, etc;
- 4) the C(P)SIG configures the (P)SIG's Event Forwarding Discriminator (EFD) table with event forwarding information such as IP address, notification type (confirmed/unconfirmed), filtering (i.e. CAS_id);
- 5) the (P)SIG generates a trigger and populates the Trigger table as configured;
- 6) the generation of a trigger causes an event to be generated, which is forwarded as defined in the EFD table;
- 7) the C(P)SIG receives the event and reads the trigger from the (P)SIG.

If the trigger causes the C(P)SIG to insert descriptors the C(P)SIG writes descriptors into the (P)SIG's descriptor insert table.

8.4.2.2 (P)SI Table Provisioning

The C(P)SIG requests (P)SI through the (P)SIG's Provisioning table. This table is just an interface to the (P)SI information within the (P)SIG or a proxy function to such information. The reply to the provisioning request contains the desired table or a part of the desired table if the table is too large to be sent in one reply. If the part number returned is 0 the table part returned is the last or only table part. Otherwise, it is the sequence number of the next table part that should be retrieved by the C(P)SIG and the C(P)SIG has to continue requesting table parts until it receives one with the sequence number 0.

8.4.2.3 (P)SI Descriptor Insertion

The C(P)SIG requests (P)SI descriptor insertion through the (P)SIG's Descriptor Insert table. That table may also be just an interface to (P)SIG's descriptor insert application. The reply to the descriptor insert request informs the C(P)SIG whether the descriptor insertion was successful.

8.4.2.4 Transport Stream Service Changes

The (P)SIG announces changes in services on the transport stream by changing a variable in the (P)SIG Configuration Table that contains the service identifiers of all services on the transport stream.

Each C(P)SIG configures the (P)SIG's event and EFD tables so that it will receive a notification if the (P)SIG transport stream service list variable changes.

8.4.2.5 PID Provisioning

The C(P)SIG requests from the (P)SIG the PID value assigned by the head-end to a stream (ECM, EMM or private data) through the PID Provisioning table. That table may be also just an interface to (P)SIG's database. The stream is identified by its type, the identifier of the CA system and the CA sub-system the stream belongs to, and a unique stream number. The reply to this request contains the PID value.

8.4.3 SIM (P)SIG Group Specification

This group consists of nine tables:

- (P)SIG Information Table;
- (P)SIG Configuration Table;
- ECM Trigger Table;
- Event Trigger Table;
- Flow PID Change Trigger Table;
- Descriptor Insert Table,
- Descriptor Insert Descriptor Table;
- Table Request Table;
- PID Provisioning Table.

8.4.3.1 Information Table

The first table is the (P)SIG information table. It is used by the (P)SIG to advertise its configuration, the transport streams it handles, and the services contained in all the transport streams. The table is indexed by simPsigIndex. The information consists of the following.

Table 38: CIM - SIM (P)SIG Group - (P)SIG Information Table

Object	Size/Description	Object Justification	Head-end/CAS Manager Maximum Access Right
simPsigIndex	2 uimsbf/unique index of the (P)SIG Table	identifies the (P)SIG communications profile	read
simPsigType	1 uimsbf/(P)SIG type definition	identifies whether the (P)SIG is a PSIG, a SIG or a PSISIG	read
simPsigTriggerSupport	4 uimsbf/trigger type definition	identifies which trigger types the (P)SIG supports	read
simPsigNetworkId	2 uimsbf/the network identifier (see EN 300 468 [1])	identifies the network	read
simPsigONetworkId	2 uimsbf/the original network identifier (see EN 300 468 [1])	identifies the original network	read
simPsigTransStreamId	2 uimsbf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read
simPsigTSServices	bslbf/list of services identifiers	identifies all services on the transport stream	read

8.4.3.2 Configuration Table

The second table is the configuration table. It is used to configure (P)SIG/C(P)SIG interaction. Each table row access is controlled by the administrative state variable and the row status variables as defined in the corresponding ITU-T and IETF standards. The table is indexed by the simPsigConfigCustCasId, the simPsigConfigIndex and the simPsigIndex. Each C(P)SIG enters one or more entries (rows) into this table. Each row defines one (P)SIG/C(P)SIG communications profile.

Each communications profile is identified by the unique index. There may be multiple entries for the same Custom Conditional Access System Identifier (simPsigConfigCustCasId). Each entry defines the types of triggers that the C(P)SIG enables and the amount of time the C(P)SIG needs after receiving a trigger to insert descriptors into the (P)SIG's table (simPsigConfigMaxCompTime). The enabling is also characterized by any combination of the following parameters: service identifier and CA_descriptor insertion mode.

Table 39 summarizes this information.

Table 39: CIM - SIM (P)SIG Group - Configuration Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigConfigIndex	2 uimbsbf/unique index of the (P)SIG Configuration Table	identifies the C(P)SIG communications profile.	read-create
simPsigIndex	2 uimbsbf/unique index of the (P)SIG Table	identifies the (P)SIG communications profile	read-create
simPsigConfigAdminState	enumerated/administrative state of the table row (as in ITU-T Recommendation X.731 [4])	enables locking for synchronized access of multiple head-end or CAS network managers	read-create
simPsigConfigCpsigType	1 uimbsbf/C(P)SIG type definition	identifies whether the C(P)SIG is a CPSIG, a CSIG or CPSISIG	read-create
simPsigConfigCustCasId	4 uimbsbf/Custom CAS Identifier	enables identification of C(P)SIGs	read-create
simPsigConfigMaxCompTime	2 uimbsbf/maximum time needed to process trigger by C(P)SIG	enables (P)SIG to estimate the time between its triggering and descriptor insertion by the C(P)SIG	read-create
simPsigConfigServiceId	2 uimbsbf/service identifier (see EN 300 468 [1])	identifies the service	read-create
simPsigConfigTriggerEnable	2 uimbsbf/trigger type definition	identifies which trigger types the C(P)SIG wants	read-create
simPsigConfigCADInsMode	enumerated/defines CA_descriptor insertion mode	identifies whether the (P)SIG has to insert the skeleton of the CA_descriptor	read-create
simPsigConfigEntryStatus	enumerated/row creation status as defined in the RFC 1901 [11] to RFC 1908 [18]	enables row creation control	read-create

8.4.3.3 ECM Trigger Table

The third table is the ECM Trigger table. It is used for the (P)SIG to enter ECM Triggers and for the C(P)SIG to read them. Upon entering a new entry (ECM Trigger) into this table an event is automatically generated by the (P)SIG as specified in the Events Configuration Table of the Simulcrypt Events Module (SEM). This event is an object value change event and carries the instance identifier of ECM Trigger Table Index as changed object identifier and the Custom CAS identifier as specific problems. This allows EFD filtering in the SEM on the Custom CAS identifier and communicates the event type (through the event name) and the ECM Trigger index to the C(P)SIG through the EFD and through a value change notification. The value change notification can be either confirmed or unconfirmed.

Each trigger is identified by the unique index (simPsigEcmTrIndex) and announces a single ECM stream creation or ECM stream deletion. The ECM stream created/deleted can be DVB Service wide or just component wide. If it is component wide a head-end wide Elementary Stream identifier is included in the table.

There may be multiple table rows per trigger depending on whether there are multiple components attached or not. Each table row is uniquely indexed by the simPsigEcmTrIndex.

Each table row consists of a network identifier (simPsigEcmTrNetworkId), the original network identifier (simPsigEcmTrONetworkId), the transport stream identifier (simPsigEcmTrTransStreamId), the service identifier (simPsigEcmTrServiceId), possibly the elementary stream identifier if the ECM stream is to be component level only (simPsigEcmTrEsId), the ECM trigger type which could be ECM stream open, close or access_criteria change (simPsigEcmTrType), the Super CAS Identifier (simPsigEcmTrSuCasId), the ECM unique identifier (simPsigEcmTrEcmId), the ECM PID (simPsigEcmTrEcmPid), the access criteria (simPsigEcmTrAccessCriteria).

Table 40 summarizes this information.

Table 40: CIM - SIM (P)SIG Group - ECM Trigger Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigEcmTrIndex	2 uimbsf/unique identifier of the ECM Trigger	identifies the (P)SIG ECM Trigger	read-create
simPsigEcmTrNetworkId	2 uimbsf/the network identifier (see EN 300 468 [1])	identifies the network	read
simPsigEcmTrONetworkId	2 uimbsf/the original network identifier (see EN 300 468 [1])	identifies the original network	read
simPsigEcmTrTransStreamId	2 uimbsf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read
simPsigEcmTrServiceId	2 uimbsf/the service identifier (see EN 300 468 [1])	identifies the service	read
simPsigEcmTrEsId	2 uimbsf/the elementary stream identifier	identifies the elementary stream if the ECMs are to be applied component wide only and not service wide	read
simPsigEcmTrType	enumerated/identifies the trigger type	distinguishes between ECM stream open, close, and access criteria change triggers	read
simPsigEcmTrSuCasId	4 uimbsf/the Super CAS identifier	identifies the CAS	read
simPsigEcmTrEcmId	2 uimbsf/the ECM stream identifier	identifies uniquely the ECM stream	read
simPsigEcmTrEcmPid	2 uimbsf/the ECM PID	identifies the ECM PID	read
simPsigEcmTrAccessCriteria	bslbf/the access criteria	specifies the access criteria	read

8.4.3.4 Flow PID Change Trigger Table

The fourth table is the Flow PID Change Trigger table. It is used for the (P)SIG to enter Flow PID Change Triggers and for the C(P)SIG to read them. Upon entering a new entry (Flow PID ChangeTrigger) into this table an event is automatically generated by the (P)SIG as specified in the Events Configuration Table of the Simulcrypt Events Module (SEM). This event is an object value change event and carries the instance identifier of Flow PID Change Trigger Table Index as changed object identifier and the Custom CAS identifier as specific problems. This allows EFD filtering in the SEM on the Custom CAS identifier and communicates the event type (through the event name) and the Flow PID Change Trigger index to the C(P)SIG through the EFD and through a value change notification. The value change notification can be either confirmed or unconfirmed.

Each trigger is identified by the unique index (simPsigFlowTrIndex) and announces a single Flow PID change. Each table row consists of the trigger index (simPsigFlowTrIndex), the flow type (simPsigFlowTrType), the super CAS identifier (simPsigFlowTrSuCasId), the flow identifier (simPsigFlowTrFlowId), and the flow PID (simPsigFlowTrFlowPID). The table is indexed by the trigger index.

Table 41 summarizes this information.

Table 41: CIM - SIM (P)SIG Group - Flow PID Change Trigger Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigFlowTrIndex	2 uimsbf/unique identifier of the Flow Trigger	identifies the (P)SIG Flow Trigger	read-create
simPsigFlowTrType	enumerated/specifies the flow type	the flow type can be ECM, EMM or private data	read
simPsigFlowTrSuCasId	4 uimsbf/the Super CAS identifier	identifies the CAS	read
simPsigFlowTrFlowId	2 uimsbf/flow identifier	uniquely identifies the flow for a given flow type and CAS identifier	read
simPsigFlowTrFlowPID	2 uimsbf/flow PID	identifies the flow PID	read

8.4.3.5 Event Trigger Table

The fifth table is the Event Trigger table. It is used for the (P)SIG to enter Event Triggers and for the C(P)SIG to read them. Upon entering a new entry (Event Trigger) into this table an event is automatically generated by the (P)SIG as specified in the Events Configuration Table of the Simulcrypt Events Module (SEM). This event is an object value change event and carries the instance identifier of the Event Trigger Table Index as changed object identifier and the Custom CAS identifier as specific problems. This allows EFD filtering in the SEM on the Custom CAS identifier and communicates the event type (through the event name) and the Event Trigger index to the C(P)SIG through the EFD and through a value change notification. The value change notification can be either confirmed or unconfirmed.

The Event Trigger table is indexed by the unique Event Trigger Index (simPsigEvtTrIndex).

Each trigger is identified by the unique index (simPsigEvtTrIndex). Each table row consists of a network identifier (simPsigEvtTrNetworkId), the original network identifier (simPsigEvtTrONetworkId), the transport stream identifier (simPsigEvtTrTransStreamId), the identifier of the service (simPsigEvtTrServiceId), the event identifier (simPsigEvtTrEventId), the start time (simPsigEvtTrStartTime), the duration of the event (simPsigEvtTrDuration), and the event related private data (simPsigEvtTrPrivateData).

Table 42 summarizes this information.

Table 42: CIM - SIM (P)SIG Group - Event Trigger Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigEvtTrIndex	2 uimsbf/unique index of the (P)SIG EVENT TriggerTable	identifies the (P)SIG Event Trigger	read
simPsigEvtTrNetworkId	2 uimsbf/the network identifier (see EN 300 468 [1])	identifies the network	read
simPsigEvtTrONetworkId	2 uimsbf/the original network identifier (see EN 300 468 [1])	identifies the original network	read
simPsigEvtTrTransStreamId	2 uimsbf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read
simPsigEvtTrServiceId	2 uimsbf/service identifier (see EN 300 468 [1])	identifies the service	read
simPsigEvtTrEventId	2 uimsbf/event identifier (see EN 300 468 [1])	identifies the event	read
simPsigEvtTrStartTime	5 bslbf/the start time in Modified Julian Date (MJD) and Universal Time, Coordinated (UTC) formats. The Field is coded as 16 bits of LSBs of MJD, followed by six 4-bit (BCD) digits representing UTC.	Indicates the start time	read
simPsigEvtTrDuration	3 uimsbf/the event duration in hours, minutes, and seconds as six 4-bit BCD digits	indicates the duration of the event	read
simPsigEvtTrPrivateData	bslbf/variable length event related user private data	event related user private data	read

8.4.3.6 PD Trigger Table

The sixth table is the PD Trigger table. It is used for the (P)SIG to enter PD Stream Event Triggers and for the C(P)SIG to read them. Upon entering a new entry (PD Trigger) into this table an event is automatically generated by the (P)SIG as specified in the Events Configuration Table of the Simulcrypt Events Module (SEM). This event is a object value change event and carries the instance identifier of PD Trigger Table Index as changed object identifier and the Custom CAS identifier as specific problems. This allows EFD filtering in the SEM on the Custom CAS identifier and communicates the event type (through the event name) and the PD Trigger index to the C(P)SIG through the EFD and through a value change notification. The value change notification can be either confirmed or unconfirmed.

Each trigger is identified by the unique index (simPsigPdTrIndex) and announces the creation, the deletion, the in PMT location modification or the content modification of a single PD stream. The concerned PD stream can be DVB Service wide or just component wide. If it is component wide a head-end wide Elementary Stream identifier is included in the table.

There may be multiple table rows per trigger depending on whether there are multiple components attached or not. Each table row is uniquely indexed by the simPsigPdTrIndex.

Each table row consists of a network identifier (simPsigPdTrNetworkId), the original network identifier (simPsigPdTrONetworkId), the transport stream identifier (simPsigPdTrTransStreamId), the service identifier (simPsigPdTrServiceId), possibly the elementary stream identifier if the PD stream is to be component level only (simPsigPdTrEsId), the PD trigger type which could be PD stream open, close, in PMT location change or content change (simPsigPdTrType), the Super CAS Identifier (simPsigPdTrSuCasId), the PD unique identifier (simPsigPdTrEcmId), the PD PID (simPsigPdTrEcmPid), the private data (simPsigPdTrPrivateData).

Table 43a summarizes this information.

Table 43a: CIM - SIM (P)SIG Group - PD Trigger Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigPdTrIndex	2 uimsbf/unique identifier of the PD Trigger	identifies the (P)SIG PD Trigger	read-create
simPsigPdTrNetworkId	2 uimsbf/the network identifier (see EN 300 468 [1])	identifies the network	read
simPsigPdTrONetworkId	2 uimsbf/the original network identifier (see EN 300 468 [1])	identifies the original network	read
simPsigPdTrTransStreamId	2 uimsbf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read
simPsigPdTrServiceId	2 uimsbf/the service identifier (see EN 300 468 [1])	identifies the service	read
simPsigPdTrEsId	2 uimsbf/the elementary stream identifier	identifies the PD stream in the PMT	read
simPsigPdTrType	enumerated/identifies the trigger type	distinguishes between PD stream open, close, in PMT location change or content change triggers	read
simPsigPdTrSuCasId	4 uimsbf/the Super CAS identifier	identifies the CAS	read
simPsigPdTrPdId	2 uimsbf/the PD stream identifier	identifies uniquely the PD stream	read
simPsigPdTrPdPid	2 uimsbf/the PD PID	identifies the PD PID	read
simPsigPdTrPdStreamType	uimsbf/the PD PID	identifies the PD stream type	read
simPsigPdTrPrivateData	bslbf/the private data	specifies the private data	read

8.4.3.7 Descriptor Insert Table

The sixth table is the Descriptor Insert table. It is used for the C(P)SIG to enter (P)SI descriptors and for the (P)SIG to read them. Entering of a descriptor completes the cycle which was started by EMM/ECM/Event triggering. The actual descriptors to be inserted are written into the descriptor table. The descriptor insert status is communicated back in the descriptor insert reply message (i.e. each set is confirmed by a get-response in SNMP) or as a value change notification (i.e. the C(P)SIG configures events and EFDs to monitor the status object identifier instance of the descriptor table entry). Each table row access is controlled by the administrative state variable and the row status variables as defined in the corresponding ITU-T and IETF standards.

Each descriptor is identified by the unique index (simPsigDescInsIndex). Each table row consists of the index and type of the trigger that caused the descriptor insertion (simPsigDescInsTrIndex, simPsigDescInsTrType), of a descriptor insert location designator (simPsigDescInsLocationId), of a number of parameters specifying the descriptor, and also the descriptor insertion delay (simPsigDescInsDelay) and the descriptor insertion delay type (simPsigDescInsDelayType).

Table 43b summarizes this information.

Table 43b: CIM - SIM (P)SIG Group - Descriptor Insert Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigDescInsIndex	2 uimsbf/unique index of the (P)SIG Descriptor Insert Table	identifies the (P)SIG Descriptor Insert	read-create
simPsigDescInsAdminState	enumerated/administrative state of the table row (as in ITU-T Recommendation X.731 [4])	enables locking for synchronized access of multiple head-end or CAS network managers	read-create
simPsigDescInsTrIndex	2 uimsbf/unique index of the (P)SIG Trigger Table	identifies the (P)SIG Trigger associated with this descriptor insertion	read-create
simPsigDescInsTrType	enumerated/identifies the type of the trigger	indicates whether the trigger index is an ECM, or Event index	read-create
simPsigDescInsLocationId	enumerated/location identifier of the descriptor destination	identifies the table and position where the descriptor is to be inserted	read-create
simPsigDescInsNetworkId	2 uimsbf/the network identifier (see EN 300 468 [1])	identifies the network	read-create
simPsigDescInsONetworkId	2 uimsbf/the original network identifier (see EN 300 468 [1])	identifies the original network	read-create
simPsigDescInsTransStreamId	2 uimsbf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read-create
simPsigDescInsServiceId	2 uimsbf/service identifier (see EN 300 468 [1])	identifies the service	read-create
simPsigDescInsElmStreamId	2 uimsbf/elementary stream identifier (see EN 300 468 [1])	identifies the elementary stream	read-create
simPsigDescInsBouquetId	2 uimsbf/bouquet identifier (see EN 300 468 [1])	identifies the bouquet	read-create
simPsigDescInsEventId	2 uimsbf/event identifier (see EN 300 468 [1])	identifies the event	read-create
simPsigDescInsONetworkId2loop	2 uimsbf/original network identifier (see EN 300 468 [1])	identifies the original network in the 2 nd loop	read-create
simPsigDescInsNetworkIdOther	2 uimsbf/network identifier (see EN 300 468 [1])	identifies the other network	read-create
simPsigDescInsTransStreamId2OrO	2 uimsbf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream in 2 nd loop or other	read-create
simPsigDescInsDelayType	enumerated/immediate or synchronized	indicates whether the (P)SIG shall immediately insert the set of descriptors in the table or shall synchronize the insertion with a triggered event start or ECM stream modification	read-create
simPsigDescInsDelay	tcimsbf/ms	if the delay insertion type is synchronized this indicates the amount of time between the time the event is supposed to occur and the time at which the descriptors are to be inserted; a negative time indicates that the tables should be broadcast before the ECM stream modification or event start	read-create
simPsigDescPrivDataSpfier	2uimsbf (see EN 300 468 [1] and ETR 162 [i.2])	Private data specifier which is inserted into the TS by (P)SIG if possible.	read-create
simPsigDescInsEntryStatus	enumerated/row creation status as defined in the RFC 1901 [11] to RFC 1908 [18]	enables row creation control	read-create

8.4.3.8 Descriptor Insert Descriptor Table

The seventh table is an extension to the descriptor insert table. It consists of a list of descriptors (simPsigDescInsDescriptor) to be inserted and their insertion return status (simPsigDescInsDescriptorStatus). The Descriptor Insert Descriptor Table is indexed by the index of the descriptor insert table and by its own unique index. This allows multiple descriptors to be associated with the same insertion and thus also with the same trigger. Each table row access is controlled by the administrative state variable and the row status variables as defined in the corresponding ITU-T and IETF standards.

Table 44 summarizes this information.

Table 44: CIM - SIM (P)SIG Group - Descriptor Insert Descriptor Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigDescInsDescIndex	2 uimsbf/unique index of the (P)SIG Descriptor Insert Descriptor Table	identifies the (P)SIG Descriptor Insert Descriptor	read-create
simPsigDescInsDescAdminState	enumerated/administrative state of the table row (as in ITU-T Recommendation X.731 [4])	enables locking for synchronized access of multiple head-end or CAS network managers	read-create
simPsigDescInsDescriptor	bslbf/the descriptor	the descriptor to be inserted	read-create
simPsigDescInsDescriptorStatus	enumerated/the descriptor insertion status	indicates whether the descriptor has been inserted or not	read-create
simPsigDescInsDescEntryStatus	enumerated/row creation status as defined in the RFC 1901 [11] to RFC 1908 [18]	enables row creation control	read-create

8.4.3.9 Table Request Table

The eighth table is the Table Request table. It is used for the C(P)SIG to request and retrieve (P)SI tables from the (P)SIG. The table is used as an interface to the (P)SI database. Since tables retrieved may exceed the maximum message size supported by the underlying transport (i.e. SNMP) a mechanism is provided to support arbitrary length tables. The reply to the provisioning request contains the desired table or a part of the desired table if the table is too large to be sent in one reply. If the part number returned is 0 this table part returned is the last or only table part. Otherwise, it is the sequence number of the next table part that should be retrieved by the C(P)SIG and the C(P)SIG has to continue requesting table parts until it receives one with the sequence number 0.

Each provisioning request is identified by the unique index (simPsigTblProvIndex). Each table row consists of a table identifier (simPsigTblProvTableId), of a network identifier (simPsigTblNetworkId), of a original network identifier (simPsigTblONetworkId), the transport stream identifier (simPsigTblTransStreamId), the service identifier (simPsigTblServiceId), the bouquet identifier (simPsigTblBouquetId), the event identifier (simPsigTblEventId), the segment number (simPsigTblSegmentNr), the table part requested (simPsigTblProvPart), and the table part number (simPsigTblProvPartNumber). The initial request always has a part number 0. Each subsequent request of the same table has the part number requested.

Table 45 summarizes this information.

Table 45: CIM - SIM (P)SIG Group - Table Request Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigTblProvIndex	2 uimbsf/unique index of the (P)SIG Table Request Table	identifies the (P)SIG Table Request	read-create
simPsigTblProvTableId	enumerated/table identifier	identifies the table requested	read-create
simPsigTblNetworkId	2 uimbsf/the network identifier (see EN 300 468 [1])	identifies the network	read-create
simPsigTblONetworkId	2 uimbsf/the original network identifier (see EN 300 468 [1])	identifies the original network	read-create
simPsigTblTransStreamId	2 uimbsf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream	read-create
simPsigTblServiceId	2 uimbsf/service identifier (see EN 300 468 [1])	identifies the service	read-create
simPsigTblBouquetId	2 uimbsf/bouquet identifier (see EN 300 468 [1])	identifies the bouquet	read-create
simPsigTblEventId	2 uimbsf/event identifier (see EN 300 468 [1])	identifies the event	read-create
simPsigTblONetworkId2loop	2 uimbsf/original network identifier (see EN 300 468 [1])	identifies the original network in the 2 nd loop	read-create
simPsigTblNetworkIdOther	2 uimbsf/network identifier (see EN 300 468 [1])	identifies the other network	read-create
simPsigTblTransStreamId2OrO	2 uimbsf/transport stream identifier (see EN 300 468 [1])	identifies the transport stream in 2 nd loop or other	read-create
simPsigTblSegmentNr	2 uimbsf/segment number(see TR 101 211 [i.3])	number of the segment	read-create
simPsigTblProvPart	bslbf/the table part	the table part	read
simPsigTblProvPartNumber	2 uimbsf/the table part number	the table part number	read-create

8.4.3.10 PID Provisioning Table

The ninth table is the PID Provisioning table. It is used for the C(P)SIG to request from the (P)SIG the PID value assigned by the head-end to a stream (ECM, EMM, or private data). The stream is identified by its type, the identifier of the CA system and the CA sub-system the stream belongs to, and a unique stream number. The reply to this request contains the PID value.

The PID Provisioning table is indexed by the Super CAS identifier (simPsigProvSuCasId), and the flow identifier (simPsigProvFlowId). A PID provisioning request is made by the C(P)SIG by simply reading the corresponding PID variable in the table, i.e. an SNMP get on simPsigProvFlowPID indexed by the right flow type, Super CAS identifier, and flow identifier. The actual implementation of the table may be just an interface to an appropriate database or application.

Table 46 summarizes this information.

Table 46: CIM - SIM (P)SIG Group - PID Provisioning Table

Object	Size/Description	Object Justification	Head-end/CA Manager Maximum Access Right
simPsigPIDProvFlowType	enumerated/flow type specification	identifies whether the PID provisioning request is for an ECM, EMM, or private data flow	read-create
simPsigPIDProvSuCasId	4 uimsbf/Super CAS identifier	identifies the CAS	read
simPsigPIDProvFlowId	2 uimsbf/flow identifier	uniquely identifies the flow for a given flow type and CAS identifier	read
simPsigPIDProvFlowPID	2 uimsbf/flow PID	identifies the flow PID	read

8.4.4 Conformance Requirements

Table 47 summarizes the conformance requirements for management entities implementing the Simulcrypt Identification Module (SIM), by group.

Table 47: CIM - SIM (P)SIG Group - Conformance Requirements

Common Information Model - CIM Simulcrypt Identification Module Group	Management Entity Hosting or Representing an ECMG		Management Entity Hosting or Representing an EMMG		Management Entity Hosting or Representing a PDG		Management Entity Hosting or Representing a (P)SIG		Management Entity Hosting or Representing a C(P)SIG	
	mndt	optnl	mndt	optnl	mndt	optnl	mndt	optnl	mndt	optnl
Ident Group	X		X		X		X		X	
ECMG Group	X									
EMMG/PDG Group (without LAPG table)			X		X					
EMMG/PDG Group (LAPG Table)				X		X				
PSIG Group							X			
CPSI Group									X	

9 (P)SIG \Leftrightarrow MUX interface

9.1 Overview

The diagram in figure 17, illustrates the exact role of the (P)SIG \Leftrightarrow MUX interface in the context of a Simulcrypt head-end system.

In this clause, the acronym (P)SIG stands either for a PSISIG, a unique logical component generating both PSI and SI tables, or for a pair (PSIG, SIG) i.e. two distinct unique logical components generating respectively PSI and SI tables.

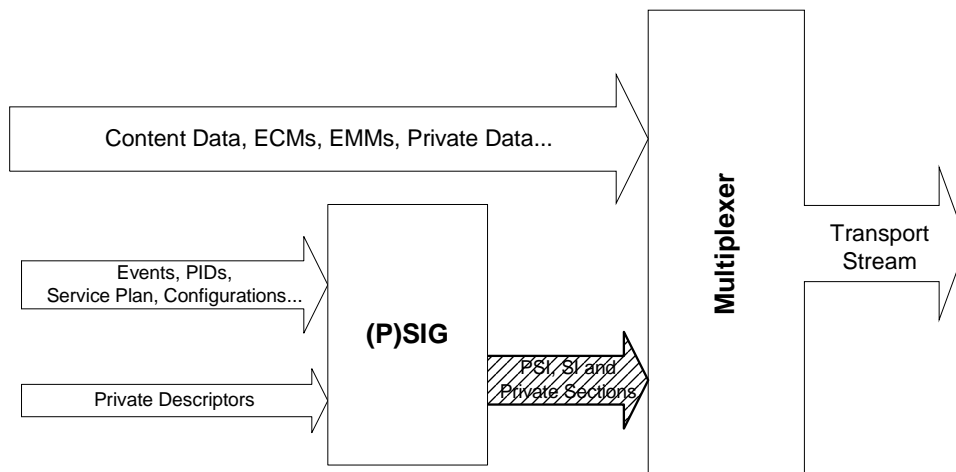
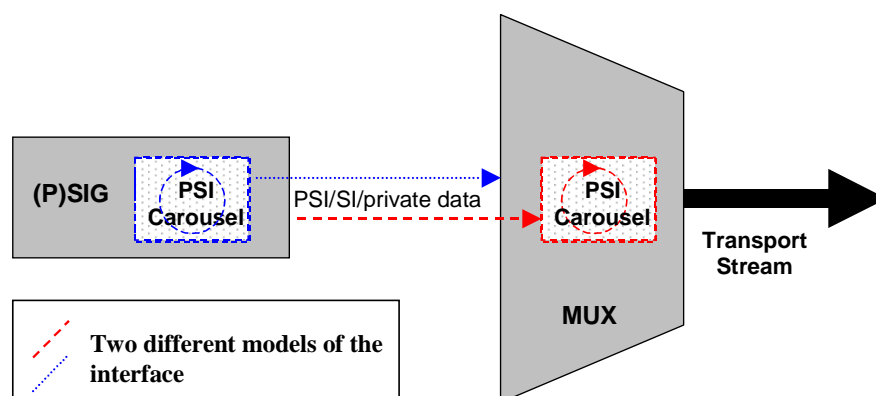


Figure 17: Role of the (P)SIG and the MUX along with their mutual relations

The main function of the (P)SIG is to provide MUXes of the head-end with the appropriate PSI/SI tables for their respective transport streams. Additionally, private sections may possibly come with the PSI/SI tables so as to be broadcast with the same PID(s).

The (P)SIG may have to deal with more than one MUX and so may have to generate specific tables for more than one transport stream. Nonetheless, each MUX gets its PSI/SI from only one unique (P)SIG.



NOTE: The location of the PSI Carousel is shown, in a logical manner, as possibly in the (P)SIG or the MUX. For each link (P)SIG \Leftrightarrow MUX of the head end, it shall be present in one of both.

Figure 18: The role of the PSI carousel logical component

These tables have to be inserted cyclically in the transport stream with respect to a specific period. The component responsible for this data carousel may be either the (P)SIG or the MUX. Two models are specified for this interface depending on which component is in charge of this data carousel (see figure 18). The choice between these models may for example be motivated according to the respective type of the different tables which are to be broadcast.

9.2 Interface principles

9.2.1 Description

This interface is based on a Channel/Stream model as used by the other connection-based protocols defined in the present document.

The (P)SIG, as a client, connects to one MUX, the server, by establishing a channel that is associated with a particular (transport_stream_ID, **original_network_ID**) pair. The (P)SIG may open as many channels with the MUX for a specific transport stream as it requires, by associating the same (transport_stream_ID, **original_network_ID**) pair with each channel. Each channel is dedicated to either of the two models of the interface: a parameter indicates at channel level if the PSI/SI tables transmitted on this channel are expected to be carouselled in the MUX or in the (P)SIG.

After the channel is open between the (P)SIG and the MUX, the (P)SIG can open streams on this channel. Each of them shall actually address one and only one PID in the TS. It is forbidden to have two different open streams dedicated to the same PID value of the same transport stream at the same time. If the (P)SIG tries to set up a stream with a PID value of this transport stream already in use by a stream of the same or another channel, it shall be answered by an error message.

From this point, the principles of the interface and the nature of the stream messages depend on which model of the interface was selected at the channel level. For this reason, the two cases are described separately in clauses 9.2.1.1 and 9.2.1.2.

9.2.1.1 Model of the interface (P)SIG ⇔ MUX with the carousel built in the MUX

This model of the interface is used to transmit, from the (P)SIG to the MUX and for each needed PID, the list of sections (PSI and/or SI tables associated to possible private tables) that the MUX will have to insert cyclically in the transport stream it generates.

Accordingly, when needed, the collection of sections which have to be inserted cyclically in the TS on a particular PID is sent once on the appropriate stream (i.e. addressing this PID value) along with their respective periods of repetition and an activation time. Later, if one or more sections shall be either updated, removed or added to those currently broadcast, a new complete collection of sections is built and transmitted to the MUX so as to replace the previous one.

A special case occurs with the management of the two DVB SI tables in charge of carrying the time and date information in the transport stream i.e. the TDT and the TOT (see EN 300 468 [1]).

Actually, the TDT role is merely to convey the current UTC-time and date information. It is thus a single short section constituted of one single parameter called UTC-time. In particular, according to EN 300 468 [1], it is not possible to insert any other descriptor in this table.

The TOT carries exactly the same information plus possible *local_time_offset_descriptors*, each indicating the local time offset of a particular zone with respect to the UTC time.

Now, the accuracy requirements in the management of these two specific tables, mainly in the population of the *UTC_time* field, impose distinct requirement for them compared with the other more *static* kinds of PSI/SI tables. In particular, it is preferable to leave the responsibility of populating the UTC time field to the MUX, which is better able to respect these precision constraints.

For this reason, the MUX is given the role of generating on its own the TDT table and of inserting it in the transport stream. Actually, the operation is to update its *UTC_time* field. The insertion period shall however respect requirement given in EN 300 468 [1] and not be greater than 30 seconds.

Regarding the TOT, instead of sending the whole table, the interface between the (P)SIG and the MUX will only be used to transmit its template, that is to say a complete version of the table but with the *UTC_time* field set to an arbitrary value. Using this template, the MUX shall generate the table in merely filling its *UTC_time* field synchronously (and in an accessory manner the CRC) before sending it out.

Each time the template of the TOT changes, in particular regarding the *local_time_offset_descriptors* it contains, the (P)SIG will transmit to the MUX a new version of the template taking into account these modifications.

9.2.1.2 Model of the interface (P)SIG ⇔ MUX with the carousel built in the (P)SIG

In this model the carousel is performed in the (P)SIG and the interface is only used to transmit the data to the MUX (in a MPEG section or transport packet format) so as to be directly inserted in the transport stream. The role of the (P)SIG in this model is very akin to the role of a Private Data Generator (PDG). The stream messages related to this model of the interface are thus based on those specified for the TCP-connection-based version of the EMMG/PDG ⇔ MUX interface (see clause 6) but with the following modifications:

- The data to transmit being only PSI/SI tables possibly associated with private data, the *data_type* parameter is removed.
- The *data_ID* parameter being no longer useful, it is replaced in the stream messages where it occurs by the *packet_ID* value of the PID with which the PSI/SI packets are to be inserted in the TS.
- As there is only one logical (P)SIG per particular transport stream, with the acronym (P)SIG standing either for PSISIG or for one pair (PSIG, SIG), the notion of *client_ID* is removed.
- Two streams open by the (P)SIG are not allowed to use at the same time the same *packet_ID* value.
- The bandwidth negotiation is mandatory for each stream and is initiated in the *stream_setup/stream_status* messages.
- The only authorized format for the datagram is the MPEG-2 packet format,
- An alternative is proposed, when transmitting the PSI/SI/Private data streams in a transport packet format, to use the DVB ASI interface (as specified in EN 50083-9 [20] and TR 101 891 [i.5]) for the data provisioning. The selection of the provisioning mode is made in the *channel_setup* message. In this case, the TCP/IP connection is preserved and all the TCP/IP messages except the *data_provision*, are still used for control purposes. This control layer is in particular used to send to the MUX the PID value of the PSI/SI section in the incoming ASI stream (the use of the ASI provisioning mode is informatively described in annex J).

9.2.2 Channel and Stream specific messages

The interface shall carry the following channel messages:

For both models:

- *channel_setup*;
- *channel_test*;
- *channel_status*;
- *channel_close*;
- *channel_error*.

For the first model when the carousel is performed in the MUX only:

- *CIM_channel_reset*.

The interface shall carry the following stream messages:

For both models:

- *stream_setup*;
- *stream_status*;
- *stream_test*;
- *stream_error*;
- *stream_close_request*;
- *stream_close_response*.

For the first model with the carousel in the MUX only:

- *CiM_stream_section_provision*.

For the second model with the carousel in the (P)SIG only:

- *CiP_stream_BW_request*;
- *CiP_stream_BW_allocation*;
- *CiP_data_provision*.

These messages are defined further in clauses 9.5 to 9.8.

9.2.3 Channel establishment

The (P)SIG is assumed to have a prior knowledge of the mapping between the pairs {transport_stream_Id, **original_network_ID**} (each identifying one particular transport stream), and the respective pairs {IP Address, Port Number} of the MUXs.

Once the TCP connection is established, the (P)SIG sends a *channel_setup* message to the MUX containing the transport_stream_ID and the **original_network_ID** of the expected TS, as well as an indication of which model of the interface is used by this channel.

When the selected model is the one with the carousel performed in the (P)SIG, the message specifies also which of either the ASI or TCP/IP provisioning mode is going to be used.

In case of success of the channel setup, the MUX replies by sending back a *channel_status* message.

In case of a rejection or a failure during channel set-up, the MUX replies with a *channel_error* message. This means that the channel has not been opened by the MUX and the (P)SIG shall close the TCP connection.

The (P)SIG can open with the same MUX as many channel for the same transport stream as it wants, and each channel is allowed to use either of the two models of the interface.

9.2.4 Stream level protocol for the model with the carousel in the MUX

9.2.4.1 Stream establishment

The (P)SIG sends a *stream_setup* message to the MUX. This message contains in particular the value of the PID with which the sections provisioned on this stream will have to be inserted in the targeted transport stream. Two different streams are not allowed to manage the same PID value at the same time.

In case of success the MUX replies by sending back a *stream_status* message.

In case of a rejection or a failure the MUX replies with a *stream_error* message. One particular case of failure may be the fact that the PID value requested is already currently used by the (P)SIG.

9.2.4.2 Provision of the PSI/SI or private sections

Once the connection, channel and stream have been correctly established, the (P)SIG can send to the MUX the sections it wants to be broadcast on the particular PID associated to this stream. The sections are transmitted in *CiM_stream_section_provision* messages and they replace the possible previous sections broadcast on this PID by the MUX.

If no section is provisioned in the *CiM_stream_section_provision* message then the MUX shall stop sending out any sections on this PID.

When the PID value associated with the stream is 0x0014, the message may then be used to send a TOT template section to the MUX in order to request of the MUX to generate the TOT. If a new template is transmitted then the MUX shall replace any possible previous version. If the *CiM_stream_section_provision* message is sent with no template then the MUX shall cease broadcasting the TOT.

9.2.4.3 Stream closure

Stream closure is always initiated by the (P)SIG. This is done by means of a *stream_close_request* message. A *stream_close_response* message indicates the stream has been closed.

However, a stream closure does not mean that the broadcasting of the associated sections is interrupted. The last previously transmitted collection of MPEG sections shall still be carouselled by the MUX so as to avoid missing (P)SI tables in the transport stream.

Nevertheless, the PID previously managed by this stream is now released. Therefore, when the (P)SIG connects to the MUX, it shall be authorized to open a stream in charge of this PID. Two situations can then occur, either the (P)SIG is still using the same model of the interface with the carousel performed in the MUX, then the MUX shall keep on sending out the previous list of tables until a new one is transmitted by the (P)SIG, or the (P)SIG now uses the other model, then the MUX shall stop the broadcast of the previous list of tables as soon as the new stream is open.

These rules apply as well in case of connection losses, channel/stream closures or channel/stream losses between the two components.

Furthermore, the (P)SIG is offered a simple way to force the MUX to stop playing out all collections of sections which are still broadcast on non-locked PIDs, namely on PIDs which are not associated to active Streams anymore. For that purpose, the (P)SIG only has to send a *CiM_channel_reset* message with an activation time.

9.2.5 Stream level protocol for the model with the carousel in the (P)SIG

9.2.5.1 Stream establishment

For this model, the (P)SIG also sends a *stream_setup* message to the MUX. In case of success the MUX replies by sending back a *stream_status* message. In case of a rejection or a failure the MUX replies with a *stream_error* message.

The *stream_setup* message contains in particular the value of the PID, called *packet_ID*, with which the PSI/SI provisioned on this stream will have to be inserted in the final transport stream. Two different streams opened by the (P)SIG are not allowed to manage the same PID value at the same time.

If the value of the *packet_ID* is 0x0014, the MUX shall cease generating the TDT and inserting it in the transport stream. As long as the stream is open, the (P)SIG is responsible for managing the TDT.

Once the connection, channel and stream have been correctly established the PSI/SI data will be transferred. They shall be transferred as TS packets. The (P)SIG indicates at *channel_setup* which of the two different data provisioning modes is selected for this channel between ASI or TCP/IP. In the ASI case, the MUX is assumed to have a prior knowledge of the physical location of the ASI stream.

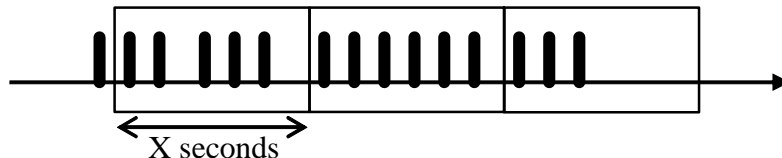
In both cases, the PSI/SI shall be inserted in the transport stream in the same order as they are provided by the (P)SIG.

9.2.5.2 Bandwidth allocation

During stream set-up the (P)SIG shall request the bandwidth needed for that stream in the *stream_setup* message. The MUX will then respond with the bandwidth that has been allocated for that stream. The (P)SIG can, at a later time, request an adjustment in the bandwidth allocation.

The (P)SIG/MUX protocol bandwidth is defined as the bandwidth occupied in the transport stream by tables provided upon the PSIG/MUX protocol, considering 188 byte TS packets (i.e. including header, = 1 504 bits per packet).

The (P)SIG/MUX protocol bandwidth is actually evaluated in a X second window applied to the transport stream. The origin of these "bandwidth windows" is arbitrary. The window length X shall be proposed by the MUX vendor and accepted by PSIG vendor.



The exceeded bandwidth error may thus be generated according to a sliding average over some bandwidth windows.

9.2.5.3 Stream closure

Stream closure is always initiated by the (P)SIG. This can occur when a PSI/SI stream is no longer needed. This is done by means of a *stream_close_request* message. A *stream_close_response* message indicates the stream has been closed.

9.2.6 Channel closure

Channel closure can occur when the Channel is no longer needed or in case of error (detected by (P)SIG or reported by MUX). This is done by means of a *channel_close* message sent by the (P)SIG. Subsequently, the connection shall be closed by both sides.

9.2.7 Channel/Stream testing and status

At any moment either component can send a *channel_test/stream_test* message to check the integrity of a channel/stream. In response to this message the receiving component shall reply with either a channel/stream status message or a channel/stream error message.

9.2.8 Unexpected communication loss

Both (P)SIG and MUX shall be able to handle an unexpected communication loss (either on the connection, channel or stream level).

Each component, when suspecting a possible communication loss (e.g. a 10 second silent period), should check the communication status by sending a test message and expecting to receive a status message. If the status message is not received in a given time (implementation specific) the communication path should be re-established.

9.2.9 Handling data inconsistencies

If the MUX detects an inconsistency it shall send an error message to the (P)SIG. If the (P)SIG receives such a message or detects an inconsistency it may close the connection. The (P)SIG (as the client) will then (re-)establish the connection, channel and (if applicable) streams.

The occurrence of a user defined or unknown parameter_type or message_type shall not be considered as an inconsistency.

9.2.10 Error management

Error processing in PSIG/MUX protocol shall be performed in a similar way as described in TR 102 035 [i.6] for all other connection-based protocols.

9.3 Parameter_type values

Table 48 defines the parameter_type values used by the interface (P)SIG \leftrightarrow MUX. The nature of each parameter is described in the following section.

Table 48: Parameter_type values for (P)SIG \leftrightarrow MUX interface

Parameter_type Value	Parameter type	Units	Length (bytes)
0x0000	DVB Reserved		
0x0001	(P)SIG_type	uimsbf	1
0x0002	channel_ID	uimsbf	2
0x0003	stream_ID	uimsbf	2
0x0004	transport_stream_ID	uimsbf	2
0x0005	original_network_ID	uimsbf	2
0x0006	packet_ID	uimsbf	2
0x0007	interface_mode_configuration	uimsbf	1
0x0008	max_stream	uimsbf	2
0x0009	table_period_pair	CompoundTLV	variable
0x000A	MPEG_section	user defined	variable
0x000B	repetition_rate	uimsbf	4
0x000C	activation_time		8
	Year	uimsbf	2
	Month	uimsbf	1
	Day	uimsbf	1
	Hour	uimsbf	1
	Minute	uimsbf	1
	Second	uimsbf	1
	Hundredth_second	uimsbf	1
0x000D	datagram	user defined	variable
0x000E	bandwidth	uimsbf/kbits/s	2
0x000F	initial_bandwidth	uimsbf/kbits/s	2
0x0010	max_comp_time	uimsbf/ms	2
0x0011	ASI_input_packet_ID	uimsbf	2
0x0012 to 0x6FFF	Reserved	-	-
0x7000	error_status	uimsbf	2
0x7001	error_information	user defined	variable
0x7003 to 0x7FFF	Reserved	-	-
0x8000 to 0xFFFF	User defined	-	-

9.4 Parameter semantics

activation_time: This integer specifies the exact time when the message shall be executed by the MUX. This value is in coordinated universal time (UTC) format i.e.:

Year		Month	Day	Hour	Minute	Second	Hundredth of seconds
MSB	LSB						
1byte	1byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

ASI_input_packet_ID: This optional parameter is only used when the PSI/SI/Private data streams are transmitting to the MUX via the ASI interface i.e. when the *interface_mode_configuration* parameter in the *channel_setup* message was previously set to 0x81. In this case, this parameter indicates to the MUX the PID value of the input PSI/SI/data streams that has to be remultiplexed in the final transport stream with the PID value given by the *packet_ID* parameter.

bandwidth: This parameter is used in *stream_BW_request* and *Stream_BW_allocation* messages to indicate the requested bit rate or the allocated bit rate respectively. It is the responsibility of the (P)SIG to maintain the bit rate generated within the limits negotiated with the MUX. It should be noted that the (P)SIG may operate from 0 kbits/s up to the negotiated rate. The (P)SIG shall not exceed the negotiated rate.

channel_ID: This identifier uniquely identifies a channel between the (P)SIG and a MUX.

datagram: This is the PSI/SI data. The Datagram shall be transferred in TS packet format only.

error_information: This parameter shall provide additional information to indicate the error condition.

error_status: This parameter shall provide a unique identifier to indicate the error condition.

initial_bandwidth: This parameter indicates for a particular stream, depending if it comes from the (P)SIG or from the MUX, the initial value in the Transport Stream either of the maximum bandwidth needed by the (P)SIG or of the bandwidth allocated by the MUX for the insertion of the PSI/SI tables which are going to be transmitted on this stream. Afterwards, the (P)SIG may request an adjustment of this value in sending a new *CiP_Stream_BW_request* message.

interface_mode_configuration: This parameter provides configuration information to the MUX. It specifies in particular which model of the interface is used by the channel. If the selected model is the one with the carousel in the (P)SIG, it also signals if the ASI provisioning mode is going to be used or not. Accordingly, the following table summarizes the possible values of this parameter.

Interface_mode_configuration values	Model of the interface	Datagram provisioning mode
0x00	Carousel of the PSI/SI/data tables performed in the MUX	in TCP/IP <i>data_provision</i> messages.
0x01 to 0x07F	Reserved	
0x80	Carousel of the PSI/SI/data tables performed in the (P)SIG	in TCP/IP <i>data_provision</i> messages.
0x81		via an ASI connection.
0x82 to 0xFF	Reserved	

max_comp_time: This parameter is communicated by the MUX to the (P)SIG during channel set-up. It is the worst case time needed by an MUX to take into account a new *CiM_stream_section_provision* message. This time is typically used by the (P)SIG to decide how long in advance the message shall be transmitted to the MUX so as to meet the expected *activation_time*.

max_stream: This integer gives the maximum number of streams a MUX is able to treat in the same time for this model of the interface.

MPEG_section: MPEG-2 section (PSI, SI and/or private section) that has to be broadcast in the transport stream addressed by the Channel with the PID value assigned to this Stream.

original_network_ID: This parameter, in combination with the *transport_stream_ID* parameter, shall uniquely identify the MPEG transport stream within the network. Refer to EN 300 468 [1] for more information.

packet_ID: This parameter indicates in the final transport stream the value of the MPEG packet identifier which will carry the sections provisioned on this stream.

(P)SIG_type: This parameter identifies if the channel is open by a PSISIG, a PSIG or a SIG according to the following table:

(P)SIG_type values	(P)SIG processes
0x01	PSIG
0x02	SIG
0x03	PSISIG

repetition_rate: This value gives in milliseconds the period of the cyclic insertion of the list of sections associated with it in a *table_period_pair* item.

stream_ID: This identifier uniquely identifies a stream within a channel.

table_period_pair: This parameter associates together the list of sections making up a table that is to be cyclically broadcast with the repetition rate at which the MUX shall achieve this cyclic insertion.

transport_stream_ID: This parameter, in combination with the **original_network_ID** parameter, shall uniquely identify the MPEG transport stream within the network. Refer to EN 300 468 [1] for more information.

9.5 Channel specific Messages

9.5.1 Channel_setup message: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1
transport_stream_ID	1
network_ID	1
(P)SIG_type	1
interface_mode_configuration	1

The *channel_setup* message (message_type = 0x0411) is sent by the (P)SIG to set up a channel once the TCP connection has been established. It shall contain the (P)SIG_type to identify the client, as well as the transport_stream_id and **original_network_id** parameters to identify the transport stream for which the channel is intended. It shall also indicate with the appropriate value of the *interface_mode_configuration* what model of the interface is used by this channel, possibly which format is used by the datagrams and which provisioning mode is selected.

9.5.2 channel_test message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1

A *channel_test* message (0x0412) may be sent by either the (P)SIG or the MUX to check if the connection is still alive and error free.

The peer shall respond with a *channel_status* message if the Channel is free of errors, or a *channel_error* message if not. If the peer does not respond within a reasonable amount of time, the sender may assume the connection is no longer valid and it may close the connection.

9.5.3 channel_status message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1
(P)SIG_type	1
interface_mode_configuration	1
max_stream	1
max_comp_time	0 or 1

The *channel_status* message (message_type = 0x 0413) is sent in response to a *channel_setup* message or *channel_test* message.

When the message is a response to a set-up, the value of the parameter *max_stream* is the one requested by the MUX. It will be valid during the whole life time of the channel. The values of the other parameters are those of the parameters provided in the related *channel_setup* message.

When the message is a response to a test, the values of all the parameters shall be those currently valid in the channel.

The *max_com_time* parameter shall be used only if the *interface_mode_configuration* parameter in the *channel_setup* message was set to 0x00. In this case, it is mandatory.

9.5.4 channel_close message: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1

A *channel_close* message (message_type = 0x 0414) shall be sent by the (P)SIG to indicate that the channel is to be closed.

No more messages shall be exchanged between the MUX and the (P)SIG.

9.5.5 channel_error message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1
error_status	1 to n
error_information	0 to n

A *channel_error* message (message_type = 0x 0415) is sent by the recipient of a *channel_setup* message or by the MUX at any time to indicate that an unrecoverable channel level error occurred. A table of possible error conditions can be found in clause 9.9.

9.6 Stream specific messages for both models

9.6.1 stream_setup message: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
packet_ID	1
ASI_input_packet_ID	0 or 1
initial_bandwidth	0 or 1

The *stream_setup* message (message_type = 0x0421) is sent by the (P)SIG to set-up a stream once the Channel has been established.

The *ASI_input_packet_ID* parameter shall be used and is mandatory only if the *interface_mode_configuration* parameter in the *channel_setup* message was set to 0x81.

The *initial_bandwidth* parameter shall be used only if the *interface_mode_configuration* parameter in the *channel_setup* message was set to 0x80 or 0x81. In this case, it is mandatory.

9.6.2 Stream_test message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1

The *stream_test* message (message_type = 0x 0422) is used to request a *stream_status* message.

The *stream_test* message can be sent at any moment by either entity. The peer shall reply with a *stream_status* message if the stream is free of errors, or a *stream_error* message if errors have occurred.

9.6.3 Stream_status message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
packet_ID	1
ASI_input_packet_ID	0 or 1
initial_bandwidth	0 or 1

The *stream_status* message is a reply to the *stream_setup* message, the *stream_test* message, or in the situation where the carousel is performed in the MUX, the *CiM_stream_section_provision* message.

The value of the *packet_ID* parameter shall be the one sent initially by the (P)SIG in the *stream_setup* message.

The values of the parameters shall be those currently valid in the stream. The *ASI_input_packet_ID* parameter is provided only if the *interface_mode_configuration* parameter in the *channel_setup* message was set to 0x81.

The *initial_bandwidth* parameter shall be used only if the *interface_mode_configuration* parameter in the *channel_setup* message was set to 0x80, or 0x81. In this case, it is mandatory. When this message is a reply to a *stream_test_message*, the returned value is the last negotiated bandwidth value currently used.

9.6.4 Stream_close_request message: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1

The *stream_close_request* message (message_type = 0x 0424) is sent by the (P)SIG to indicate that the stream is to be closed.

9.6.5 Stream_close_response message: (P)SIG \Leftarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1

The *stream_close_response* message (message_type = 0x 0425) is sent by the MUX indicating the stream that is being closed.

9.6.6 Stream_error message: (P)SIG \Leftrightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
error_status	1 to n
error_information	0 to n

A *stream_error* message (message_type = 0x0426) is sent by the recipient of a *stream_test* message or by the MUX at any time to indicate that an unrecoverable stream level error occurred. A table of possible error conditions can be found in clause 9.9.

9.7 Specific messages for the model with the carousel in the MUX

The use of the following messages is allowed only if the *interface_mode_configuration* parameter in the *channel_setup* message was previously set to 0x00.

9.7.1 CiM_stream_section_provision: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
activation_time	1
table_period_pair	0 to n

A *CiM_stream_section_provision* message (message_type = 0x 0431) is sent by the (P)SIG to transmit the list of tables that the MUX shall broadcast in the transport stream it manages. The MUX will have to insert cyclically and simultaneously, on the PID associated to this stream, each list of sections given in each *Table_period_pair* item with the period specified for each by the *repetition_rate* value.

The MUX shall respond with a *stream_status* or *stream_error* message. It is recommended the MUX rapidly respond.

The format of the *table_period_pair* parameter is defined as follows:

Parameter	Number of instances in message
MPEG_section	n
repetition_rate	1

The *activation_time* parameter indicates when the MUX has to start the broadcast of these sections. If it indicates a moment in the past, the broadcast is to be started immediately or as soon as possible.

The broadcast by the MUX of all these transmitted sections replaces the one of those possibly previously provisioned on this PID. If no *table_period_pair* item is supplied, it means that no section has to be sent out on this PID anymore.

If the value of the *packet_ID* associated with this stream is 0x0014, a TOT template may be supplied in the *MPEG_section* parameter of one of the transmitted *table_period_pair* items. In that case the format of the template shall comply with the one specified in EN 300 468 [1] and the value of the *table_ID* shall be 0x73. The MUX shall then use this template to generate the TOT by overwriting synchronously its *UTC_time* field (and also the CRC). Afterward, it shall insert it in the transport stream according to the *repetition_rate* parameter specified in the *table_period_pair* item.

9.7.2 CiM_channel_reset: (P)SIG \Rightarrow MUX

Parameter	Number of instances in message
channel_ID	1
activation_time	1

A *CiM_channel_reset* message (message_type = 0x0432) is sent by the (P)SIG to the MUX so as to force the latter to stop playing out all the previously transmitted collection of PSI/SI/private sections currently broadcast on PIDs not locked anymore by a stream.

The *activation_time* parameter indicates when the MUX has to perform this reset. If it indicates a time in the past, the broadcast is to be stopped immediately or as soon as possible.

9.8 Specific messages for the model with the carousel in the (P)SIG

The use of the following messages is allowed only if the *interface_mode_configuration* parameter in the *channel_setup* message was previously set to 0x80 or to 0x81.

9.8.1 CiP_Stream_BW_request message: (P)SIG ⇒ MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
bandwidth	0 or 1

The *CiP_stream_BW_request* message (message_type = 0x 0441) is always sent by the (P)SIG and can be used in two ways.

If the bandwidth parameter is present the message is a request for the indicated amount of bandwidth.

If the bandwidth parameter is not present the message is just a request for information about the currently allocated bandwidth.

The MUX shall always reply to this message with a *stream_BW_allocation* message.

9.8.2 CiP_stream_BW_allocation message: (P)SIG ⇐ MUX

Parameter	Number of instances in message
channel_ID	1
stream_ID	1
bandwidth	1

The *CiP_stream_BW_allocation* message (message_type = 0x0442) is used to inform the (P)SIG about the bandwidth allocated. This is always a response to a *CiP_stream_BW_request* message. The message is always sent by the MUX.

9.8.3 CiP_stream_data_provision message: (P)SIG ⇒ MUX

Parameter	Number of instances in message
channel_ID	0 to 1
stream_ID	0 to 1
packet_ID	1
datagram	1 to n

The *CiP_stream_data_provision* message (message_type = 0x0443) is used by the (P)SIG to send, on a given *stream_ID*, the datagram containing the PSI/SI/Private data to insert in the Transport stream with a PID value given by the *packet_ID* parameter.

This message is not used in case of ASI provisioning of the PSI/SI/data streams.

9.9 Error status

NOTE: TCP connection level errors are beyond the scope of the present document. Only channel, stream and application level errors are specified. These errors occur during the life time of a TCP connection.

There are two different error messages on these interfaces. The *channel_error* message for channel wide errors and the *stream_error* message for stream specific errors. These messages are sent by the MUX to the (P)SIG.

When the (P)SIG reports an error to the MUX, it is up to the MUX to decide the most appropriate step to be taken. However "unrecoverable error" explicitly means that the channel or stream (depending on the message used) has to be closed. Most of the error status listed in the table below can not occur in normal operation. They are mainly provided to facilitate the integration and debugging phase.

When the mention *CiM error type* (respectively *CiP error type*) appears in the description of an error type, the related *error_status* value shall be employed only when the carousel is performed in the MUX (respectively in the (P)SiG).

Table 49: (P)SIG ↔ MUX protocol error values

Error_status value	Error type
0x0000	Reserved
0x0001	Invalid message
0x0002	Unsupported protocol version
0x0003	Unknown message_type value
0x0004	Message too long
0x0005	Unknown stream_ID value
0x0006	Unknown channel_ID value
0x0007	Too many channels on this MUX
0x0008	Too many data streams on this channel
0x0009	Too many data streams on this MUX
0x000A	Unknown parameter_type
0x000B	Inconsistent length for parameter
0x000C	Missing mandatory parameter
0x000D	Invalid value for parameter
0x000E	Inconsistent value of transport_stream_ID
0x000F	Inconsistent value of packet_ID
0x0010	channel_ID value already in use
0x0011	stream_ID value already in use
0x0012	Stream already open for this pair (transport_stream_ID, packet_ID)
0x0013	Overflow error when receiving the list of MPEG sections (CiM error type)
0x0014	Inconsistent format of TOT template (CiM error type)
0x0015	Warning: Difficulties in respecting the requested repetition_rates for the last 5 minutes (CiM error type)
0x0016	Warning: Difficulties in respecting the requested Bandwidth for the last 5 minutes (CiP error type)
0x0017 to 0x6FFF	Reserved
0x7000	Unknown error
0x7001	Unrecoverable error
0x7002 to 0x7FFF	DVB Reserved
0x8000 to 0xFFFF	User Defined

10 EIS ↔ SCS Interface

10.1 Overview

This clause defines how the Event Information Scheduler (EIS) shall communicate with the Simulcrypt Synchronizer (SCS) to provide Scrambling Control Group (SCG) definitions and access criteria transitions knowing that the SCGs of an SCS shall be configured by one and only one EIS.

The EIS has many functions, one of which is to provide the SCS with the SCGs, each defining the service(s) and/or elementary stream(s) to be scrambled using a common Control Word (CW) key. The EIS also manages each CA system for each Entitlement Control Message (ECM) stream and schedules access criteria transition for each stream on the SCS. The SCS combines the access criteria with each new CW and requests an ECM from Entitlement Control Message Generator. For each Crypto Period (CP), the SCS sends the CW to the scrambler and the ECM(s) to the multiplexer for broadcast.

According to the interface between the ECMG and the SCS as specified in clause 5, *ECM_ID* is passed between the SCS and the ECMG to uniquely identify an ECM stream within a Super CAS ID for the entire system. The *ECM_ID* decouples the ECM stream from the actual ECM PID and allows the CAS to abstract itself from the MPEG-2 transport level.

In addition to the ECM stream, the EIS must also provide the SCS with a definition of the content elementary streams to be scrambled within the SCG. The obvious solution is to use the *service_ID* as defined in EN 300 468 [1]. But DVB does not restrict all elementary stream within a program to be scrambled with the same CW key. To utilize this flexibility, it may be necessary for the EIS to specify the logical identifier of one particular component instead of the *service_ID*. For this reason, the SCS may allow SCG definitions to contain *component_ID* (logical identifier referencing a particular elementary stream i.e. a particular component of a service) level definitions in addition to *Service_ID*.

Nevertheless it is not the scope of the present document to specify the exact mechanisms used by the SCS and/or the MUX to get the information needed to perform the mapping between these logical identifiers such as the *component_ID* or the *service_id*, and MPEG-2 transport level references, namely the PIDs of the related elementary streams in the TS. This is deemed as being implementation dependant. As an example, the SCS and/or the MUX could get this information from the MUXCONFIG knowing that the interface between the EIS and the MUXCONFIG would be used to make the link from a logical service plan to the related physical resources. But, these different protocols are out of the scope of the present document.

The SCS has a prior knowledge of the mapping between *super_CAS_IDs* and the IP addresses and port numbers of the ECMGs. When a new ECM stream is requested by the EIS for a given *super_CAS_ID* value, the SCS will open a new stream with the appropriate ECMG. This might require the opening of a new channel (which involves the opening of a new TCP connection).

10.2 Interface principles

10.2.1 Channel specific messages

This interface shall support the following channel messages:

- *channel_setup;*
- *channel_test;*
- *channel_status;*
- *channel_close;*
- *channel_reset;*
- *channel_error.*

10.2.2 Scrambling Control Group (SCG) specific messages

This interface shall support the following SCG messages:

- *SCG_provision;*
- *SCG_test;*
- *SCG_status;*
- *SCG_error;*
- *SCG_list_request;*
- *SCG_list_response.*

For this interface, the EIS is the client and the SCS is the server.

The EIS has prior knowledge of the mapping between each SCS, its IP address and port number, and the transport(s) controlled by the SCS.

There can be several SCSs within the network, each controlling a subset of the transport space in the system. Additionally, more than one SCS may be configured for the same transport to provide redundancy.

10.2.3 Channel establishment

There is always one (and only one) channel per TCP connection. Once the TCP connection is established, the EIS sends a *channel_setup* message to the SCS.

In the case of success, the SCS replies with a *channel_status* message.

In the case of a rejection or failure during channel set-up the SCS replies with a *channel_error* message. This means that the channel is not established and the EIS shall close the TCP connection.

10.2.4 Scrambling Control Group provisioning

Based on the *channel_status* message received from the Simulcrypt Synchronizer (SCS) during channel establishment, the EIS can provision the SCS with the proper SCG definitions.

There are three modes supported by the SCS:

- service level definitions;
- elementary stream (*component_ID*) level definitions;
- mix of service level and ES level definitions.

If the SCS supports only service level definitions, the EIS shall only use the *service_ID* in the *SCG_provision* message to define groups.

If the SCS supports only elementary stream level definitions, the EIS shall only use the *component_ID* in the *SCG_provision* message to define groups.

If the SCS supports both service level definitions and ES level definitions, the EIS may use either *service_ID* or *component_ID* to define groups.

In all cases, the EIS must ensure that the definition of an SCG active at a certain time does not conflict with the definition of any other SCGs active at the same time.

10.2.5 Channel closure

Channel closure can occur when the channel is no longer needed or in case of error (detected by EIS or reported by SCS). This is done by means of a *channel_close* message sent by the EIS. Subsequently, both sides shall close the connection.

Now, if the connection is broken between the EIS and the SCS, the SCS shall consider the Channel as closed.

Nevertheless, even if the Channel is closed or in the case when the connection between the EIS and the SCS is broken, the SCS shall keep on managing previously transmitted SCGs and preserve their respective scrambling statuses.

10.2.6 Channel testing and status

After a channel has been established, either the EIS or the SCS may send a *channel_test* message to check the integrity of a channel. In response to this message the receiving component shall reply with a *channel_status* message or *channel_error* message.

10.2.7 Scrambling Control Group testing and status

After a channel has been established, the EIS may send a *SCG_test* message to check the status of a SCG. In response to this message the SCS shall reply with a *SCG_status* message or *SCG_error* message.

10.2.8 Unexpected communication loss

Both EIS and SCS shall be able to handle an unexpected communication loss (either on the connection or channel level).

Each component, when suspecting a possible communication loss, should check the communication status by sending a test message. If a status message is not received in a given time (implementation specific) the communication path should be re-established.

10.2.9 Handling data inconsistencies

If the SCS detects an inconsistency it shall send an error message to the EIS. After receiving the error message, the EIS may close the connection. The EIS (as the client) may then re-establish the connection.

10.2.10 Error management

Error processing in EIS/SCS protocol shall be performed in a similar way as described in TR 102 035 [i.6] for all other connection-based protocols.

10.3 Parameter_type values

Table 50: EIS ↔ SCS protocol parameter_type values

Parameter_type Value	Parameter type	Units	Length (bytes)
0x0000	DVB Reserved		
0x0001	EIS_channel_ID	uimbsf	2
0x0002	service_flag	boolean	1
0x0003	component_flag	boolean	1
0x0004	max_SCG	uimbsf	2
0x0005	ECM_Group	compoundTLV	variable
0x0006	SCG_ID	uimbsf	2
0x0007	SCG_reference_ID	uimbsf	4
0x0008	Super_CAS_ID	uimbsf	4
0x0009	ECM_ID	uimbsf	2
0x000A	access_criteria	user defined	variable
0x000B	activation_time	complexTLV	8
	year	uimbsf	2
	month	uimbsf	1
	day	uimbsf	1
	hour	uimbsf	1
	minute	uimbsf	1
	second	uimbsf	1
	hundredth_second	uimbsf	1
0x000C	activation_pending_flag	boolean	1
0x000D	component_ID	uimbsf	2
0x000E	service_id	uimbsf	2
0x000F	transport_stream_ID	uimbsf	2
0x0010	AC_changed_flag	boolean	1
0x0011	SCG_current_reference_ID	uimbsf	4
0x0012	SCG_pending_reference_ID	uimbsf	4
0x0013	CP_duration_flag	uimbsf	1
0x0014	recommended_CP_duration	uimbsf/n x 100ms	2
0x0015	SCG_nominal_CP_duration	uimbsf/n x 100ms	2
0x0016	original_network_ID	uimbsf	2
0x0016 to 0x6FFF	DVB Reserved	-	-
0x7000	error_status	uimbsf	2
0x7001	error_information	user defined	variable
0x7002	error_description	ASCII byte string	variable
0x7003 to 0x7FFF	Reserved	-	-
0x8000 to 0xFFFF	User defined	-	-

10.4 Parameter Semantics

AC_changed_flag: This parameter signals a change of the Access Criteria associated to one particular ECM stream. It shall be used to indicate to the SCS to use the *AC_delay* values from an ECMG for the first CP.

access_criteria: This parameter contains CA system specific information of variable length and format, needed by the ECMG to build an ECM.

activation_pending_flag: This parameter shall indicate that the *SCG_provision* message was queued on the SCS until the activation time.

activation_time: This parameter specifies the exact time a crypto-period transition shall take place in 10 ms resolution. If it is associated with a creation or deletion of a SCG, it specifically denotes the moment when the change of the scrambling status occurs.

Year		Month	Day	Hour	Minute	Second	Hundredth of seconds
MSB	LSB						
1byte	1byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

This value is in coordinated universal time (UTC) format. For example:

- { 00 0B 00 08 07 CF 03 1B 0F 1E 00 3C } = March 27, 1999 15:30:00 600 ms

The manner this parameter can be used is informatively illustrated in annex G.

component_flag: This parameter shall indicate if the SCS supports SCG definitions based on *component_IDs*. If set to TRUE, the EIS may send *SCG_provision* messages with *Component_IDs* as content resources.

component_ID: This parameter uniquely identifies an elementary stream within the transport stream identified by the *transport_stream_ID* parameter. This abstracts the EIS from knowing the transport level details of the system.

CP_duration_flag: This parameter shall indicate if the SCS supports the recommended crypto-period duration value provisioning or not.

- "FALSE" indicates that the SCS is not able to process any crypto-period values it receives.
- "TRUE" indicates that the SCS is able to process the recommended crypto-period value associated to each SCG.

ECM_Group: This parameter shall contain all the information to bind an ECM stream to a CA provider. This information includes *super_CAS_ID*, *ECM_ID*, and *access_criteria*.

ECM_ID: This parameter shall uniquely identify an ECM stream within a CA system. This abstracts the CA system from knowing the transport level details of the system.

EIS_channel_ID: This parameter shall uniquely identify the TCP connection between the EIS and the SCS.

error_description: This parameter shall provide an ASCII text string to assist in the description of the error condition.

error_information: This parameter shall provide additional information to indicate the error condition.

error_status: This parameter shall provide a unique identifier to indicate the error condition.

max_SCG: This parameter shall identify the total number of SCG definitions supported on the SCS.

original_network_ID: This parameter, in combination with the *transport_stream_ID* parameter, shall uniquely identify the MPEG transport stream within the network. Refer to EN 300 468 [1] for more information.

recommended_CP_duration: This parameter indicates the value of the crypto-period duration desired by the EIS for the associated SCG.

service_ID: This parameter shall uniquely identify the DVB service within the transport as specified in EN 300 468 [1].

service_flag: This parameter shall indicate if the SCS supports SCG definitions based on *service_IDs*. If set to TRUE, the EIS may send *SCG_provision_messages* with *service_IDs* as content resources.

SCG_ID: This parameter shall uniquely identify a Scrambling Control Group (SCG) within the system. Each SCG may contain a collection of programs or elementary streams and one or more ECM group. The *SCG_ID* is used by both the EIS and the SCS to reference the SCG.

SCG_reference_ID: This parameter shall provide the EIS with a mechanism to uniquely identify the provisioning for a SCG group. This value shall not be modified by the SCS for any purpose. The exact semantic of this identifier is EIS implementation dependant and can be ignored by the SCS.

SCG_current_reference_ID: The *SCG_reference_ID* associated with the currently active SCG.

SCG_nominal_CP_duration: The *SCG_nominal_CP_duration* indicates the nominal Crypto_Period duration set by the SCS for a particular SCG.

SCG_pending_reference_ID: This *SCG_reference_ID* associated with the queued SCG.

super_CAS_ID: This parameter shall consist of the *CAS_ID* and a *sub-ID*. It binds an *ECM_ID* to a particular ECMG on the SCS. Refer to ETR 162 [i.2] for more information.

transport_stream_ID: This parameter, in combination with the *original_network_ID* parameter, shall uniquely identify the MPEG transport stream within the network. Refer to EN 300 468 [1] for more information.

10.5 Channel specific messages

Each message denotes the direction the message may be passed:

- EIS⇒SCS - EIS may send this message to SCS only;
- EIS⇐SCS - SCS may send this message to EIS only;
- EIS⇔SCS - This message may be sent by either EIS or SCS.

10.5.1 channel_setup message: EIS ⇒ SCS

Parameter	Number of instances in message
EIS_channel_ID	1

The *channel_setup* message (message_type = 0x0401) is sent from the EIS to the SCS when the connection is first opened. Once a channel has been established, this message shall not be sent.

The SCS shall respond with a *channel_status* message or, in the case of *EIS_Channel_ID* already being in use, with a *channel_error* message.

10.5.2 channel_test message: EIS ⇔ SCS

Parameter	Number of instances in message
EIS_channel_ID	1

A *channel_test* message (message_type = 0x0402) may be sent by either the EIS or SCS to check if the connection is still alive and error free.

The peer shall respond with a *channel_status* message if the channel is free of errors, or a *channel_error* message. If the peer does not respond within a reasonable amount of time, the sender may assume the connection is no longer valid and it may close the connection.

10.5.3 channel_status message: EIS ⇔ SCS

Parameter	Number of instances in message
EIS_channel_ID	1
service_flag	1
component_flag	1
max_SCG	1
CP_duration_flag	1

The *channel_status* message (message_type = 0x0403) is sent in response to a *channel_setup* message, *channel_test* message or a *channel_reset* message.

This message defines the channel parameters between the EIS and the SCS.

If the SCS supports service level SCG definitions, *service_flag* shall be set to TRUE.

If the SCS supports Elementary Stream (ES) level SCG definitions, *component_flag* shall be set to TRUE.

The SCS may support both service level SCG definitions and ES level SCG definitions simultaneously.

The *CP_duration_flag* indicates if the SCS is able to process the recommended crypto-period values in the *SCG_provision* message.

10.5.4 channel_close message: EIS \Rightarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1

A *channel_close* message (message_type = 0x0404) shall be sent by the EIS to indicate the channel is to be closed.

No more messages shall be exchanged between the SCS and the EIS.

10.5.5 channel_reset message: EIS \Rightarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1
transport_stream_ID	0 to n
activation_time	0 or 1

A *channel_reset* message (message_type = 0x0406) may be sent by the EIS to perform SCG-deprovisioning of all the previous provisions sent to the SCS. SCS shall consider all the SCG deprovisioned and the SCS may remove them all from its internal bookkeeping. In addition, all the components of these deprovisioned SCG shall not be scrambled anymore.

If no *transport_stream_ID* parameter is present, this message concerns all the SCGs currently managed by the SCS.

If *transport_stream_ID* values are provided, this message concerns only the SCGs related to one of the transport stream identified by one of these values.

If *activation_time* is present, the deprovisioning shall be triggered at the *activation_time*. If no *activation_time* is provided the deprovisioning will be performed immediately.

Until a new SCG_provision message is sent to the SCS by the EIS, the SCS shall return no *SCG_ID* in the *SCG_list_response* message responding to a possible *SCG_list_request* message.

The SCS shall respond with a *channel_status* message if the channel is free of errors, or a *channel_error* message.

10.5.6 channel_error message: EIS \Leftrightarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1
error_status	1 to n
error_information	0 to n
error_description	0 to n

A *channel_error* message (message_type = 0x0405) is sent by the recipient of a *channel_test* message or by the SCS at any time to indicate that an unrecoverable channel level error occurred. A table to possible error conditions can be found in clause 10.7.

10.6 SCG specific messages

10.6.1 SCG_provision message: EIS \Rightarrow SCS

Parameter	Number of instances in message
<i>EIS_channel_ID</i>	1
<i>SCG_ID</i>	1
<i>transport_stream_ID</i>	1
<i>network_ID</i>	1
<i>SCG_reference_ID</i>	0 or 1
<i>activation_time</i>	0 or 1
<i>recommended_CP_duration</i>	0 or 1
<i>component_ID</i>	0 to n
<i>service_ID</i>	0 to n
<i>ECM_Group</i>	0 to n

A *SCG_provision* message (message_type = 0x0408) is sent by the EIS to the SCS to create, modify or de-provisioned a SCG:

- when the SCG identified in the message is created, the concerned services or components are scrambled and associated ECM are generated and broadcast;
- when the SCG is modified, new services or components in this SCG are scrambled and associated to current ECM(s); deleted services or components in this SCG are not scrambled anymore;
- when the SCG identified in the message is de-provisioned, the concerned services or components are not scrambled anymore and generation and broadcast of associated ECM are stopped.

The SCS shall respond by a *SCG_status* message or a *SCG_error* message. The response is sent immediately by the SCS even if the *SCG_provision* message refers to an *activation_time*.

When several *SCG_provision* messages refer to different SCG but at the same *activation_time*, the SCS shall decide for an error occurring in the definition of the SCGs only after the activation time.

Each SCG provisioning message shall contain a *transport_stream_ID* and an **original_network_ID** to identify the reference of the output transport stream for which the content resources are defined.

The *SCG_reference_ID* may contain an identifier used by the EIS to uniquely reference a provisioning group. This value shall not be modified by the SCS for any purpose. The EIS may choose to use it as a unique identifier across the whole system or within an *SCG_ID*. For instance, a usual utilization for the EIS of this identifier may be to make the distinction between two different versions of the same SCG.

If *activation_time* is present, the SCS shall queue the provision message until the specified activation time. Only one SCG provision message for a particular *SCG_ID* may be queued at a time. If another message is queued when a new message is posted with or without *activation_time* parameter, the new message will replace the old message in the queue.

The EIS shall send a provision message containing an *activation_time* to the SCS at least one CP duration before a scheduled transition. This allows the SCS to regenerate some or all of the ECMs. In that case, the SCS may extend a Crypto-Period (CP) to achieve the desired activation time. But it shall never shorten a CP to hit an activation time.

If the EIS sends a provision message with an *activation_time* less than one CP duration, the SCS shall return an error notifying the EIS that the requested *activation_time* could not be achieved. The SCS shall however accept the provisioning message.

If the EIS sends a *SCG_provision* message with an *activation_time* less than one CP duration or with no activation time, the SCS shall process it as soon as possible. And only in such a situation and in the case of access criteria change of existing SCG, the SCS is allowed to shorten the duration of the current CP. However, it shall never shorten it to a time lower than all the *min_CP_duration* values specified at *Channel_setup* by the ECMGs connected to the SCS and involved by this SCG.

ECM_Group is defined in clause 10.4.

If there are no content resource identifier, namely neither *component_id* nor *service_id* parameter, and no *ECM_Group* items specified in the message then the SCS shall consider the SCG deprovisioned and the SCS shall remove it from its internal bookkeeping. If *activation_time* is present, the deprovision shall be triggered at the *activation_time*. If no *activation_time* is provided the deprovisioning will be performed immediately. When an SCG has been deprovisioned, the SCS shall no longer return *SCG_ID* in response to an *SCG_list_request* message.

If the SCG contains one or more *ECM_Group* items and no content resource identifier, the SCS shall return an error and discard the provisioning message. If there was an active SCG, it shall remain in effect.

If the SCG contains one or more content resource identifiers and no *ECM_Group* items then the SCS shall return an error and discard the provisioning message. If there was an active SCG, it shall remain in effect.

If a *service_ID* is removed from an SCG, all content streams within that service shall no longer be scrambled.

If a *component_ID* is removed from an SCG, that elementary stream shall no longer be scrambled.

If an *ECM_Group* is removed from the SCG, then that ECM stream shall no longer be associated with the SCG and the *transition_delay_stop* value, **as defined by the ECMG⇌SCS protocol**, shall be used to specify the end of that ECM broadcast.

If an *ECM_Group* is added to the SCG, then that ECM stream shall use the *transition_delay_start* **defined by the ECMG⇌SCS protocol** to specify the beginning of its ECM broadcast.

If a scrambling state transition occurs, then a *transition_delay_stop* or *transition_delay_start*, **as defined by the ECMG⇌SCS protocol**, shall be used for each ECM Group within the SCG. Scrambling state transitions include both scrambled-to-clear and clear-to-scrambled transitions.

The SCS shall respond with an *SCG_status* or *SCG_error* message. It is recommended the SCS rapidly responds. If the SCS does not respond quickly enough to multiple messages the EIS requests, the TCP connection will eventually back up and delay subsequent messages from being transmitted.

A *service_ID* may only be sent from the EIS to the SCS if the SCS had set the *service_flag* in the *channel_status* message to TRUE.

A *component_ID* may only be sent from the EIS to the SCS if the SCS had set the *Component_flag* in the *channel_status* message to TRUE.

At the option of the SCS, both *service_ID* and *Component_ID* may be used in the same *SCG_provision* message. Individual elementary streams may be used by more than one program. The method of coordination is not defined.

The *recommended_CP_duration* should always be considered by the SCS as a recommendation only when setting the actual crypto-period duration of the related SCG and since the choice of this value depends eventually also on the different *min_CP_duration* values returned by each involved ECMGs. The way the SCS shall manage this value is detailed in annex H.

If the *CP_duration_flag* was set to FALSE in the *channel_status* message. The SCS shall ignore this value if provided.

10.6.2 SCG_test message: EIS ⇒ SCS

Parameter	Number of instances in message
EIS_channel_ID	1
SCG_ID	1

The *SCG_test* message (message_type = 0x0409) may be sent by the EIS to verify the status of a SCG.

The SCS shall respond with a *SCG_status* message, if the channel and SCG are error free. Otherwise an *SCG_error* message is sent.

10.6.3 SCG_status message: EIS \Leftarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1
SCG_ID	1
SCG_current_reference_ID	0/1
SCG_nominal_CP_duration	0/1
SCG_pending_reference_ID	0/1
activation_pending_flag	1

A *SCG_status* message (message_type = 0x040A) is sent by the SCS in response to a *SCG_provision*, or *SCG_test* message from the EIS in which there was no error. If there were an error, the SCS would respond with a *SCG_error* message defined in clause 10.6.6.

The *SCG_nominal_CP_duration* parameter shall be present only if an *SCG_current_reference_ID* is present. The *SCG_nominal_CP_duration* indicates the nominal crypto-period duration used by the SCS for the SCG identified by both the *SCG_id* and the *SCG_current_reference_ID* i.e. the active version of the SCG.

If the SCS does not know yet the *SCG_nominal_CP_duration* value at the time when the *SCG_status message* must be returned, the SCS may omit it.

10.6.3.1 Response to a provisioning message

The following conditions apply if the SCS is responding to a *SCG_provision* message:

- If *SCG_reference_ID* and no *activation_time* were provided in the provision message, the *SCG_current_reference_ID* value shall be set to the value received by the provision. The *SCG_nominal_CP_duration* value shall indicate the nominal CP duration allocated to this SCG.
- If both *SCG_reference_ID* and *activation_time* were provided in the provision message, *SCG_pending_reference_ID* shall be set to the value received in the provision message.
- If the provision message results in the SCS pending the provision request until some *activation_time* in the future, the *activation_pending_flag* shall be set to TRUE, otherwise it shall be set to FALSE.

10.6.3.2 Response to a test message

The following conditions apply if the SCS is responding to a *SCG_test* message:

- If *SCG_reference_ID* exists in the current SCG, *SCG_current_reference_ID* shall be set to its value. The *SCG_nominal_CP_duration* value shall indicate the nominal CP duration allocated to this SCG.
- If there is a pending SCG and it has an *SCG_reference_ID*, *SCG_pending_reference_ID* shall be set to its value.
- If there is a SCG group queued then *activation_pending_flag* shall be set to TRUE, otherwise it shall be set to FALSE.

10.6.3.3 Management of the *SCG_nominal_CP_duration* parameter

In the *SCG_status* message, via the *SCG_nominal_CP_duration* parameter, the SCS provides the value of the nominal crypto-period allocated to the currently valid SCG, namely the value of the crypto-period duration for the SCG which is active at the moment when either the *SCG_provision* message or the *SCG_test* message being replied by this *SCG_status* message is received. If no SCG is active at this moment, *SCG_nominal_CP_duration* parameter shall be missing.

Accordingly, in the case when the EIS sends an *SCG_provision* message containing an activation time, the value of the CP duration which will be possibly provided in the *SCG_status* message replied by the SCS, will not be the one allocated to the SCG given in this *SCG_provision* message but the one of the current active SCG if any.

Now, if the EIS wishes to know the actual CP duration allocated to the SCG provisioned in this *SCG_provision* message, it shall first actually wait that this new SCG becomes active, that is to say that the activation time occurs. After, it shall explicitly request to get this value of the SCS in sending a new *SCG_test* message being replied by a new *SCG_status* message where the new SCG has now become active and as a consequence, for which the SCS may now provide the requested CP duration value.

10.6.4 SCG_list_request message: EIS \Rightarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1

The *SCG_list_request* message (message_type = 0x040C) may be sent by the EIS to request the list of *SCG_IDs* active on the SCS.

The SCS shall respond with a *SCG_list_response* message if the channel is error free. Otherwise a channel error message is sent.

10.6.5 SCG_list_response message: EIS \Leftarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1
SCG_ID	0 to n

A *SCG_list_response* message (message_type = 0x040D) is sent in response to a *SCG_list_request* message. This message defines the stored *SCG_ID* values provisioned on the SCS.

Stored *SCG_IDs* are defined as having one or more resources and are either active or pending based on the SCG activation_time.

10.6.6 SCG_error message: EIS \Leftarrow SCS

Parameter	Number of instances in message
EIS_channel_ID	1
SCG_ID	1
SCG_reference_ID	0 or 1
error_status	1 to n
error_information	0 to n
error_description	0 to n

The *SCG_error* (message_type = 0x040B) message is sent by the SCS in response to a *SCG_provision*, or *SCG_test* message from the EIS where the SCG is in an error state.

A table of possible error conditions can be found in clause 10.7.

10.6.7 ECM_Group: CompoundTLV

Parameter	Number of instances in message
ECM_ID	1
Super_CAS_ID	1
access_criteria	1
ac_changed_flag	0 or 1

The *ECM_Group* contains all the necessary information to generate ECMs for a particular ECM stream.

ECM_ID is defined by DVB ETR 162 [i.2] and uniquely identifies an ECM stream. The combination of *ECM_ID* and *Super_CAS_ID* uniquely identifies each ECM stream within the whole system.

The *access_criteria* parameter is required when the access criteria content changes. If the *access_criteria* parameter is a pointer to access criteria content, e.g. a database, this parameter is required when the referenced content changes.

In an *SCG_provision* message providing a new version of an SCG, the EIS shall always set the *ac_changed_flag* of the related *ECM_groups* to TRUE if this *ECM_group* is a new one or if its access criteria have been changed.

If the *ac_changed_flag* is TRUE then the SCS shall apply the appropriate *ac_delay* values for this ECM stream for the first CP. This shall disable the SCS ability to detect a change in the access criteria.

10.7 Error status

NOTE: TCP connection level errors are beyond the scope of the present document.

There are two different error messages on this interface; the *channel_error* message and the *SCG_error* message. The *channel_error* message is used to communicate a channel-level error. The *SCG_error* message is returned specifically for SCG provisioning and includes additional information to identify the *SCG_ID*.

It is up to the SCS to determine the most appropriate step to be taken for each error. However, "unrecoverable error" explicitly means that the channel has to be closed. Most of the errors listed below can not occur in normal operation. They are mainly provided to facilitate the integration and debugging phase.

Table 51: SCS protocol error values

error_status value	Error type
0x0000	DVB Reserved
0x0001	Invalid message
0x0002	Unsupported protocol version
0x0003	Unknown message_type value
0x0004	Message too long
0x0005	Inconsistent length for parameter
0x0006	Missing mandatory parameter
0x0007	Invalid value for parameter
0x0008	Unknown <i>EIS_channel_ID</i> value
0x0009	Unknown <i>SCG_ID</i> value
0x000A	Max SCGs already defined
0x000B	Service level SCG definitions not supported
0x000C	Elementary Stream level SCG definitions not supported
0x000D	<i>Activation_time</i> possibly too soon for SCS to be accurate
0x000E	SCG definition cannot span transport boundaries
0x000F	A resource does not exist on this SCS
0x0010	A resource is already defined in an existing SCG
0x0011	SCG may not contain one or more content entries and no <i>ECM_Group</i> entries
0x0012	SCG may not contain one or more <i>ECM_Group</i> entries and no content entries
0x0013	<i>EIS_channel_ID</i> value already in use
0x0014	Unknown <i>Super_CAS_Id</i> .
0x0015 to 0x6FFF	Reserved
0x7000	Unknown error
0x7001	Unrecoverable error
0x7002 to 0x7FFF	Reserved
0x8000 to 0xFFFF	EIS specific/CA system specific/User defined

A resource is defined as a service, an elementary stream or an ECM stream.

11 ACG \leftrightarrow EIS Interface

11.1 Overview

In the context of a typical DVB Simulcrypt implementation, the way the access criteria management is perceived at the scheduling side is as follows:

- **In the *commercial world*:** Service editors, operators, and content providers transmit some information concerning the access conditions associated to the programs that are to be broadcast. This information is transmitted along with the schedule information through conductors or play lists using interfaces to the EIS. Nevertheless, since the information these providers produce is mostly "editorial" and as they usually have a low knowledge of the proprietary CAS related aspects, the nature of this conditional access information associated with the events can only be "editorial" as well (notion of product, of the involved CAS systems, of commercial profiles, etc.), even if their exact nature may be the results of an agreement with CAS providers.
- **At the head-end:** According to DVB Simulcrypt, part of the role of the EIS is to transmit access criteria to the SCS in a format that is understandable to the corresponding ECMG, so that it is able to generating ECMs.

A translation needs thus to be performed between these two worlds.

In the current version of the standard, the EIS is assumed to operate this translation from the incoming "editorial" conditional access information to AC that is comprehensible by each related ECMG. However, the format of this AC blob is CAS-proprietary whereas components such as the EIS, SCS or (P)SIG are assumed to be "neutral" in DVB Simulcrypt, i.e. CAS independent.

For this reason, a new element supplied by CAS Providers has been introduced in the DVB Simulcrypt architecture model. This new component, which directly connects to the EIS, is responsible for performing this translation. The name given to this new CAS component is *Access Criteria Generator*. Each CAS responsible for the generation of its own AC shall supply its own ACG.

11.2 Scope

The present document defines the standard interface between the EIS and the ACG. It basically allows the EIS to request access criteria from the ACG for each event involving its respective CAS. The access criteria are always related to one and only one *event*. An *event* in this clause means a scheduled change of the status or of the conditions to access a certain group of components (audio, video, data, etc.) broadcast within one of the transport streams generated by the head-end.

11.2.1 Role of the ACG

In the EIS \leftrightarrow ACG interface, the ACG takes on the role of the *server*. The ACG is responsible for providing the EIS with all CAS-proprietary information related to each event. This information is then transmitted by the EIS to SCS and possibly also to the (P)SIG.

The ACG is specifically responsible for providing the EIS with access criteria data, which will be sent by the SCS to the ECMG in the *access_criteria* parameter of the *CW_provision* message of the ECMG protocol. The content and the meaning of specific access criteria data is proprietary to each CAS. It is assumed that this content is not modified by the EIS or by the SCS while it is on route from the ACG to the ECMG.

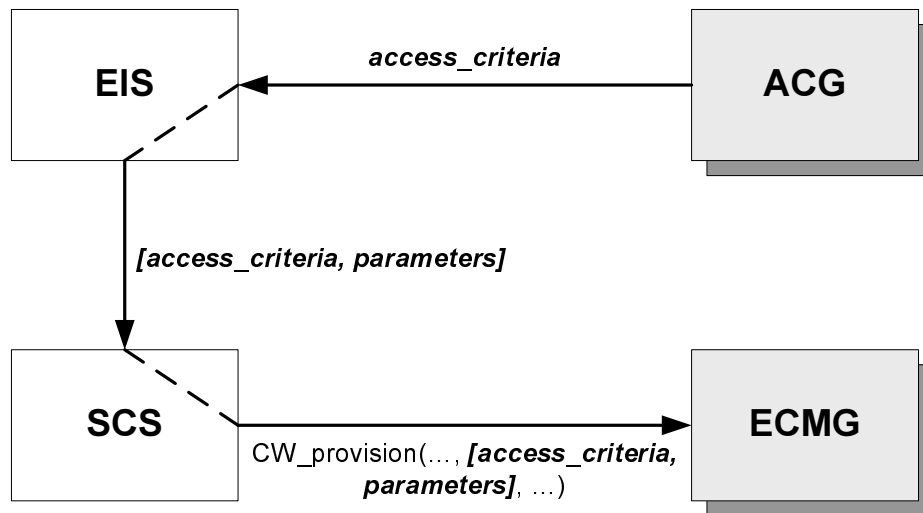


Figure 19: Flow of access criteria from ACG to ECMG

The ACG transmits the access criteria to the EIS after receiving an appropriate request from the EIS. The AC can be accompanied by one or more CA private data *blobs* that are meant to be transmitted to the related C(P)SIG via the (P)SIG synchronously to their transmission to the ECMG via the SCS. Both access criteria and CA data *blobs* are related to the same event.

An additional mechanism is offered along with these access criteria, to allow the ACG to request that the EIS add other event-related parameters to the access criteria before the latter transmits them to the SCS.

In the response containing the access criteria, the ACG is able to indicate an associated expiration time after which the related access criteria need to be updated. If this AC is in use after the AC expiration time, the EIS shall request updated AC at the specified AC expiration time. The ACG is responsible for the management of the delay between this expiration time and the time needed to transmit new AC to the ECMG.

In addition, the ACG may also notify the EIS more explicitly that it needs to modify the access criteria already associated with a particular event. Typically, the ACG shall do that sufficiently in advance before the AC change really needs to occur.

11.2.2 Role of the EIS

In the EIS⇔ACG interface, the EIS takes on the role of the *client*. The underlying assumption is that the EIS does not have to include any CAS specific knowledge and that it is the unique owner of the event schedule information in the head-end.

The EIS is responsible for scheduling the transmission of event-related data to other DVB Simulcrypt head-end components, and for scrambling of the TS components. Scrambled to non-scrambled transition is done according to EIS⇔SCS protocol (see clause 10). In particular, the EIS manages the synchronization of the changes related to a specific schedule event that needs to be executed at the same time in the SI and PSI tables, and in the ECMs containing the access criteria.

Figure 20 summarizes the data flow exchanges that take place around the EIS. Note that the boxes with shadows are provided by the CAS vendors. It is also important to observe that since the ACG, ECMG and C(P)SIG are furnished by the CAS, links may exist between these components that are not represented on figure 20.

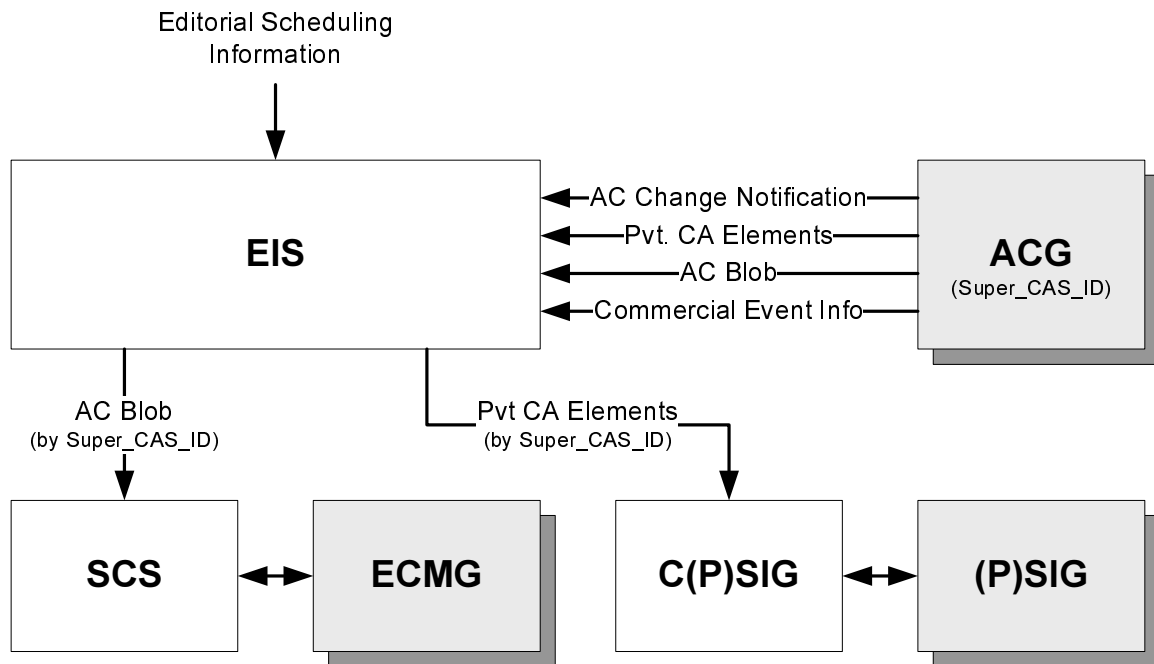


Figure 20: Flow of data to/from the EIS

Whenever a new event is about to start, the EIS is required to send a message to the SCS. This message, specified by the EIS⇌SCS protocol, is a request for the generation of new related ECM streams or for the update of existing ECM streams. Each of the messages shall contain the associated *super_CAS_id* and the AC Blob containing CAS-specific access criteria.

The C(P)SIG may in parallel request the PSIG (and optionally the SIG) for the simultaneous insertion/update of private or CA descriptors in PSI and SI tables.

As a consequence, the EIS sends requests before an event occurs to the ACGs that are associated with the event, so as to get the AC sets for each ECM stream that is to be played out for this event. CA private data may be associated with the AC, which is transmitted to each associated C(P)SIG via the (P)SIG. This private CA data can be transferred to the C(P)SIG as a part of the *event_related_data*, using the *private_data* parameter.

It is assumed that all the requests coming from the EIS are made sufficiently in advance of the start of an event, so as to cover all the constraints raised by the synchronization of the different head-end equipment impacted by this change.

The EIS shall deal with the expiration time that may optionally be associated with the access criteria values received from the ACG.

The EIS may also be notified by the ACG that previously transmitted AC needs to be updated for undisclosed (CAS proprietary) reasons.

11.2.3 Dynamics of the ACG⇌EIS Interface

The ACG⇌EIS protocol is based on the *on-request* model. The ACG provides an access criteria value on request from the EIS or on its own for updating a previously transmitted AC value. The EIS shall in any case send as many requests to an ACG as there are events relating to that CAS (identified by its *Super_CAS_ID*) and in return may receive as many AC values. This means that any AC value is associated with one and only one event.

For each event, each request shall contain a selection of related commercial event information. The information is specific to each CAS. The nature of the commercial event information is based on an agreement between the event information provider and each CAS and will not be specified by DVB.

This on-request model is depicted in figure 21.

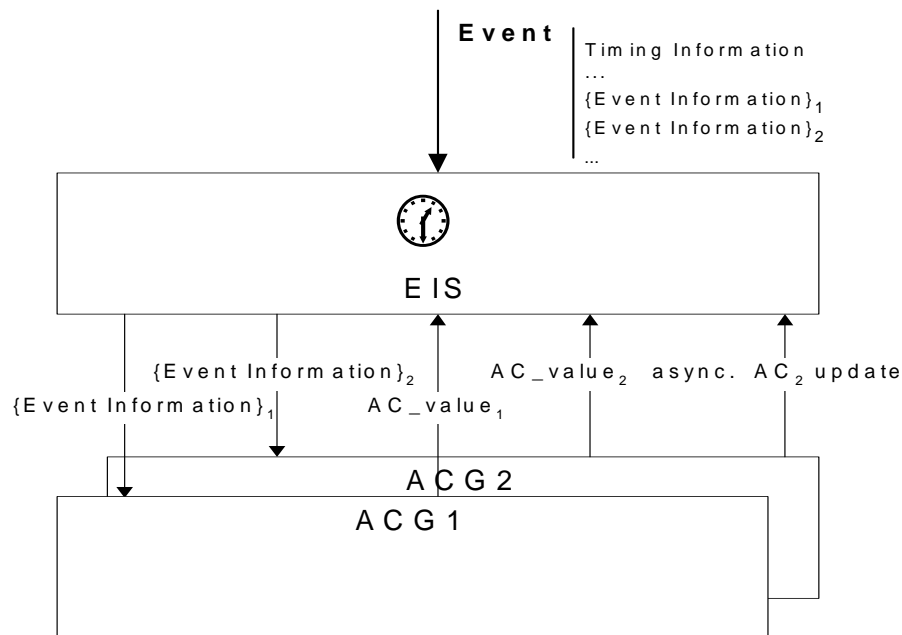


Figure 21: The *On-Request* Model

Optionally, the interface offers the ACG a way to notify that (for CAS-proprietary reasons) previously transmitted AC needs to be updated. In this case, a referencing mechanism is defined to allow the ACG to identify the AC value that is required to be updated.

11.3 Interface Principles

11.3.1 General Principles

This protocol has the following characteristics:

- It is a connection-oriented TCP protocol.
- The message structures in the protocol are XML-based.
- An XML Schema Definition (XSD) is used for defining the protocol structure.

11.3.2 Channel Specific Messages

This interface carries the following channel messages:

- *channel_setup*;
- *channel_test*;
- *channel_status*;
- *channel_close*;
- *channel_error*.

As previously stated, for the purpose of the EIS⇌ACG interface the EIS is the client and the ACG is the server. The EIS has prior knowledge of the mapping between *Super_CAS_ID* and the IP address and port number of each ACG.

NOTE: There can be several ACGs associated with each *Super_CAS_ID* value (e.g. for performance or redundancy reasons). In such a case, the EIS should be able to choose with which ACG the connection will be opened, based on either a redundancy policy or resource availability.

11.3.3 Messages for Access Criteria Creation and Modification

This interface carries the following channel messages for access criteria creation and modification:

- *ac_request*;
- *ac_response*;
- *ac_interrupt*;
- *ac_error*.

11.3.4 Channel Establishment

There is always one (and only one) channel per TCP connection. Once the TCP connection is established the EIS sends a *channel_setup* message to the ACG. In case of success, the ACG replies by sending back a *channel_status* message.

In the case of a rejection or a failure during channel setup, the ACG replies with a *channel_error* message. This means that the channel has not been opened by the ACG, and the EIS shall close the TCP connection.

11.3.5 Channel Closure

Channel closure can occur when the channel is no longer needed or in the case of an error (detected by the EIS or reported by the ACG). This is done by means of a *channel_close* message, which is sent by the EIS to the ACG. Subsequently, the connection shall be closed by both sides.

11.3.6 Channel Testing and Status

Either component may send a *channel_test* message at any time to check the integrity of a channel. In response to this message the receiving component shall reply with either a *channel_status* message or a *channel_error* message.

11.3.7 Unexpected Communication Loss

Both EIS and ACG shall be able to handle an unexpected communication loss (either on connection or channel level).

When suspecting a possible communication loss (e.g. a 10 second silent period) each component should check the communication status by sending a test message and expecting to receive a status message. If the status message is not received within a given time (which is implementation specific) the communication path should be re-established.

11.3.8 Handling Data Inconsistencies

If the ACG detects an inconsistency it shall send an error message to the EIS. If the EIS receives such a message or detects an inconsistency it may close the connection. The EIS (as the client) will then (re-)establish the connection and the channel.

The occurrence of a user-defined or otherwise unknown *parameter_type* or *message_type* shall not be considered as an inconsistency.

11.3.9 Handling Multiple AC Parameters in one AC Response

Each CAS may need to generate several distinct ECM streams for the same event, depending on the CAS vendor's implementation of different business models. The number of ECM streams required for an event managed by a specific CAS is therefore decided by the CAS. The latter communicates it to the EIS in the AC response message, which it sends as a result of the AC request it gets for this event. The EIS shall be able to infer this number by counting the instances of the *ACG_access_criteria* structure present in the message.

Subsequently, the EIS shall create as many *ECM_groups* as indicated by this number and associate each with one of the *ACG_access_criteria* blocks provided in these *ACG_access_criteria* structures. It shall also allocate a specific *ECM_id* to each of these ECM groups. Finally, it shall gather and send all these ECM groups together to the SCS in the SCG related to the event.

11.3.10 Handling AC Expiration Time

The ACG is able to indicate an associated expiration time in the response message to the EIS that contains the access criteria. This expiration time indicates the time after which the access criteria need to be updated if still in use. The ACG defines this time by using the *ac_expiration_time* parameter. If the event end time is later than this AC expiration time, then the EIS must request new AC for this event when this AC expiration time occurs. In this case the EIS must send an *ac_request* message to the ACG to receive new AC for this event.

11.3.11 Asynchronous Access Criteria Change Request

As described in clause 11.2.3, the ACG provides one or more sets of access criteria in response to a commercial event information provisioning command from the EIS. Each access criteria set is identified by an *ac_id* parameter that is provided and managed by the ACG. The ACG may request the EIS to update a previous access criteria set by asynchronously sending a new access criteria value. The access criteria set that the EIS is required to update is identified by the *ac_id* parameter.

The *ac_interrupt* message is sent by the ACG to make the EIS apply a new set of parameter values for a particular *ac_id*. The message semantic is as follows:

ac_interrupt (*ac_id*, *access_criteria*, [*ac_parameter*],
[*ac_expiration_time*], [*private_CA_element*])

This is depicted in figure 22.

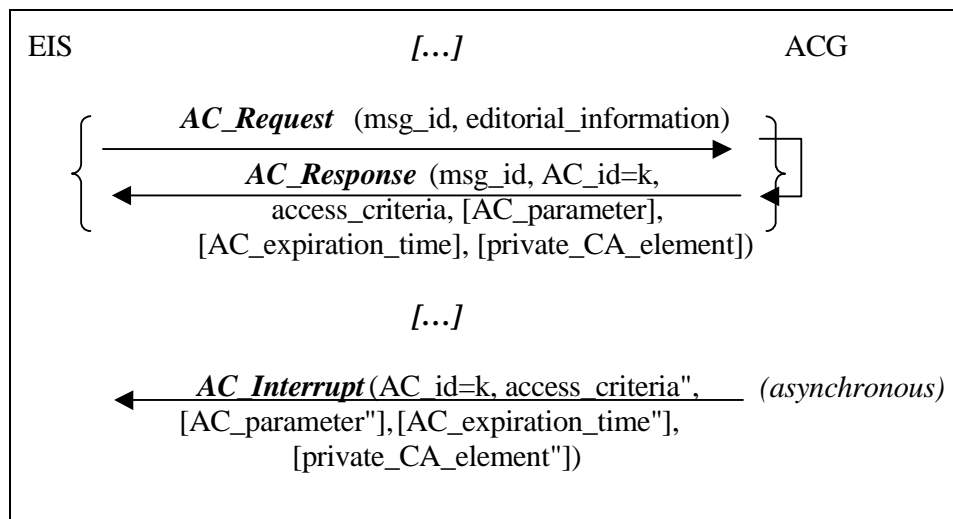


Figure 22: The AC Interrupt Message

The *ac_id* parameter is always assigned by the ACG:

- The *ac_id* is allocated by the ACG in order to identify the access criteria it sends in response to an commercial event information from the EIS.
- The ACG is free to choose the *ac_id* allocation policy, including the mechanism for re-use of previous values.
- The *ac_id* is referenced by the ACG for updating previous access criteria.

The *ac_id* parameter is supported by the EIS only for allowing the update of a previous access criteria:

- The EIS stores an *ac_id* value as long as the relevant access criteria is "significant" in the scheduling process, including play list editing and play-out. The definition of this significance is left up to the EIS to decide, and may including the decision to ignore the *ac_id* parameter.

- When receiving an *ac_response* message:
 - the EIS cancels the previous *ac_id/access* criteria link for the same *ac_id* value, if any
 - and the EIS stores the new *ac_id/access* criteria link;
 - the above actions are not performed if the EIS ignores the *ac_id* parameter.
- When receiving an *ac_interrupt* message:
 - if the EIS knows this *ac_id* value, the EIS updates the access criteria;
 - if the EIS does not know the *ac_id* value, the EIS does not modify anything and sends an error message back to the ACG.

11.3.12 Inserting Additional Information in the AC

When sending the access criteria to the EIS, the ACG may request the EIS to add other event-related information to the AC (e.g. start time of the event), before the latter transmits the AC to the SCS. To notify the EIS, the ACG puts the list of *ac_parameters* at the end of the AC. Each parameter in this list is authenticated by an *ac_parameter_tag*. The actual value of *ac_parameter* is not known to the ACG, and the EIS must substitute the *ac_parameter* with the actual value of the appropriate parameter before the AC will be sent to the SCS (refer to clause 11.7 for a detailed description).

Support for the *ac_parameter* is optional. If the EIS supports this mode it signals the ACG by using the *parameter_flag* in the *channel_setup* message. If the ACG supports this mode it signals this fact to the EIS by using the *parameter_flag* in the *channel_status* message.

11.3.13 Data Communications and Message Format

All messages exchanged between the SIMCOMP and the MUXCONFIG share a common general format as detailed in clause 4.4.2. The messages are encoded using XML over a TCP connection. The XML Schema Definition for this protocol is described in annex M.

11.4 Interface Structure

The ACG⇌EIS interface has the follows high-level structure.

Table 52: General Interface Structure

Diagram	
Children	msg_header EIS_message ACG_message
Source	<pre> <xs:element name="EIS_ACG_message"> <xs:complexType> <xs:sequence> <xs:element name="msg_header" type="message_header_type"/> <xs:choice> <xs:element name="EIS_message" type="EIS_message_type"/> <xs:element name="ACG_message" type="ACG_message_type"/> </xs:choice> </xs:sequence> </xs:complexType> </xs:element> </pre>

The *message_header* and the *message_header_type* have the following structures:

Table 53: The *msg_header* element

Diagram	msg_header			
Type	message_header_type			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	protocol_version	protocol_version_type	Required	
	ACG_channel_ID	channel_ID_type	Required	
Source	<xs:element name="msg_header" type="message_header_type"/>			

Table 54: The *message_header_type* Complex Type

Diagram	message_header_type			
Used by	element	EIS_ACG_message/msg_header		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	protocol_version	protocol_version_type	Required	
	ACG_channel_ID	channel_ID_type	Required	
Source	<xs:complexType name="message_header_type"> <xs:attribute name="protocol_version" type="protocol_version_type" use="required"/> <xs:attribute name="ACG_channel_ID" type="channel_ID_type" use="required"/> </xs:complexType>			

Figure 23 describes the EIS messages structure.

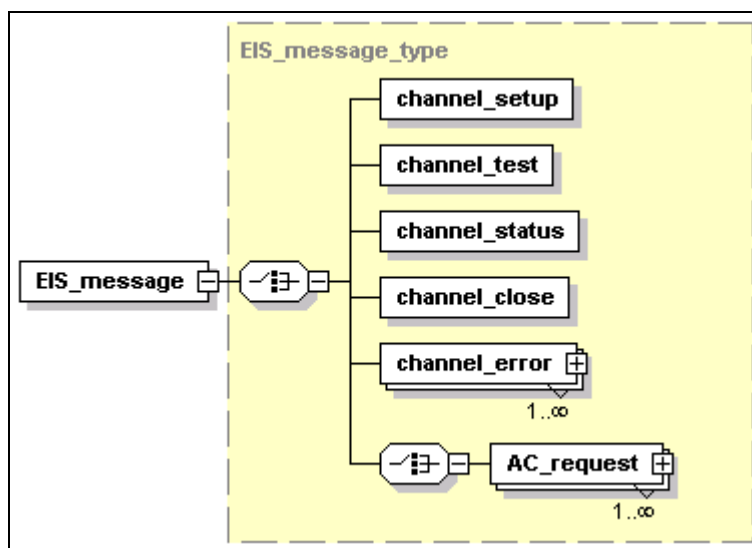


Figure 23: EIS Message Structure

Figure 24 describes the ACG messages structure.

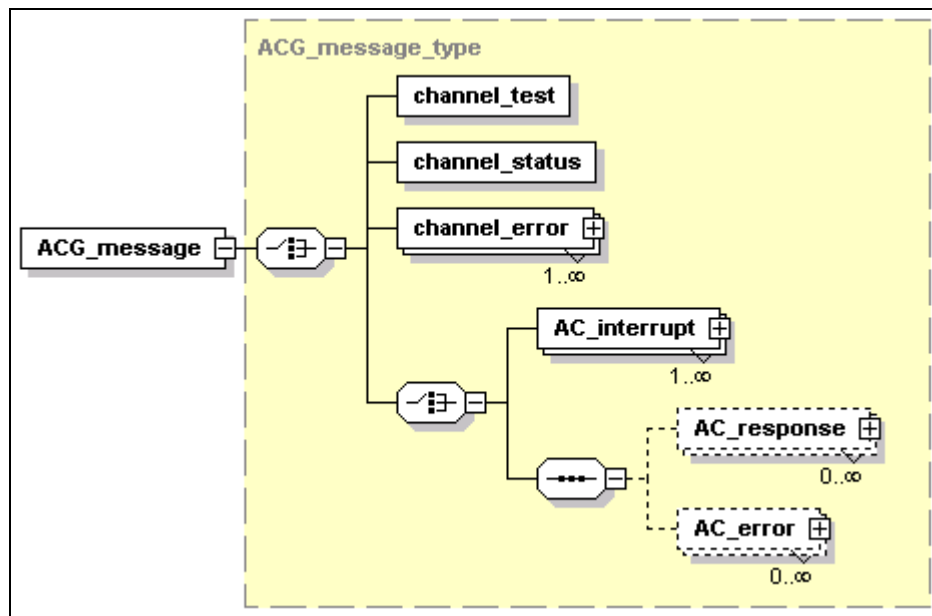


Figure 24: The ACG Message Structure

11.5 Parameter Semantics

access_criteria_struct: This is a complex parameter, which contains *ACG_access_criteria*, *ac_id*, *AC_parameter(s)*, *AC_expiration_time*, and *private_CA_element*. The *AC_response* message contains one or more such parameters. Refer to clause 11.3.9 for details.

ACG_access_criteria: This parameter represents an access criteria generated by the ACG from *commercial_event_information*. If the ACG requests the EIS to substitute the *AC_parameter(s)*, then the access criteria sent from the EIS to the SCS will consists of *ACG_access_criteria* and *AC_parameter(s)*. If the ACG does not request the EIS to substitute the parameters, then the access criteria sent by the EIS to the SCS will be equal to the *ACG_access_criteria* parameters. See detailed *AC_parameter* explanation below.

ACG_channel_ID: This parameter shall uniquely identify the TCP connection between the EIS and the ACG.

AC_expiration_time: This parameter is used by the ACG to inform the EIS, when the appropriate AC shall be updated. The EIS shall send a new *commercial_event_information_provision* message to the ACG upon this expiration.

AC_ID: This parameter is sent by the ACG to the EIS in an *AC_response* message and further is used by the ACG to reference the appropriate AC in the *AC_interrupt* message. This parameter shall be unique per *super_CAS_id*.

AC_parameter: This parameter is used to allow the ACG to request the EIS to add other event-related information to the access criteria before the EIS transmits them to the SCS. The parameter is identified by the *AC_parameter_tag*. At some later stage the EIS generates AC and includes the list of pairs: *<AC_parameter_tag, AC_parameter_value>*. The list of valid *AC_parameter_tags* and mechanism used by the EIS for parameter substitution is defined in clause 11.7.

component_ID: This parameter uniquely identifies an elementary stream within the transport stream identified by the *original_network_ID* and *transport_stream_ID* parameters. This abstracts the EIS from knowing the transport level details of the system. This parameter is used as a value in the *AC_parameter* (see annex M).

commercial_event_information: This parameter contains information used by the ACG to generate AC. This information is specific to each CAS (*super_CAS_id*).

error_information: This optional parameter contains user defined data completing the information provided by *error_status*. It can be an ASCII text or the parameter ID value of a faulty parameter for example.

error_status: Refer to clause 11.10.

event_ID: This parameter contains the identification of the event described within the EIT (uniquely allocated within a *service_ID*). Refer to the DVB SI specification (EN 300 468 [1]) for more information. This parameter is used as a value in the *AC_parameter* (see annex M).

interrupt_flag: This parameter is used by the ACG in the *channel_status* message to signal the EIS that the ACG will use *AC_interrupt* messages. In this case, the flag has the "TRUE" value within the *channel_status* message.

- This parameter is used by the EIS in the *channel_setup* message to signal the ACG that the EIS is able to serve *AC_interrupt* message. In this case, the flag has the "TRUE" value within the *channel_setup* message.
- The ACG can either use the *AC_interrupt* message or the *AC_expiration_time* parameter to ask the EIS to change AC.

max_comp_time: This parameter defines the worst case time needed by an ACG to compute an AC.

msg_ID: This parameter shall uniquely identify a message and enable asynchronous message exchange. The response to the message shall have the same *msg_id* as the request. New *msg_ids* are generated by EIS only. The range of the *msg_id* is divided in two sub-ranges:

- 0 to 32 767 for EIS command messages and associated ACG response messages.
- 32 768 to 65 535 for ACG command messages and associated EIS response messages.

original_network_id: This parameter is used in combination with the *transport_stream_ID* to uniquely identify the MPEG transport within the network. Refer to DVB SI specification (EN 300 468 [1]) for more information.

parameter_flag: This parameter is used by the ACG in the *channel_status* message to signal the EIS that the ACG will send the parameters list, in addition to the AC. In this case the flag has a "TRUE" value.

- This flag is also used in the *channel_setup* message to indicate to the ACG whether the EIS supports the *AC_parameter* mode or not. The flag should have the value "TRUE" in the case the *AC_parameter* mode is supported.

private_CA_element: This parameter contains the CAS-related data to be transmitted to (P)SIG.

private_CA_element_flag: This parameter is used by the ACG in the *channel_status* message to signal the EIS that the *private_CA_descriptor* will be sent in the *CA_response* messages. In this case, the flag has the value "TRUE".

protocol_version: This parameter identifies the protocol version. **It shall have the value 0x04 according to the present document.**

service_ID: This parameter shall uniquely identify the DVB service within the transport. This parameter is used as a value *AC_parameter*.

super_CAS_id: 32-bit identifier formed by the concatenation of the *CA_system_id* and the *CA_subsystem_id*.

transport_stream_ID: This parameter is used in combination with the *original_network_ID* to uniquely identify the MPEG transport within the network. Refer to the DVB SI specification (EN 300 468 [1]) for more information.

11.6 Parameter Types

Table 55 describes the parameter type. See more detailed definition in the XML schema in annex M.

Table 55: Parameter Type Definition

Parameter	Type Name	Type
<i>access_criteria_struct</i>	<i>access_criteria_structure_type</i>	complex type
<i>acg_access_criteria</i>	<i>acg_access_criteria_type</i>	xs:hexBinary
<i>acg_channel_id</i>	<i>channel_id_type</i>	xs:unsignedShort
<i>ac_expiration_time</i>	<i>ac_expiration_time_type</i>	xs:dateTime
<i>ac_id</i>	<i>ac_id_type</i>	xs:unsignedInt
<i>ac_parameter_tag</i>	<i>ac_parameter_tag_type</i>	xs:unsignedShort
<i>commercial_event_information</i>	<i>commercial_event_info_type</i>	Abstract
<i>error_information</i>	<i>error_information_type</i>	xs:string
<i>error_status</i>	<i>error_status_type</i>	xs:unsignedShort
<i>interrupt_flag</i>		xs:Boolean
<i>max_comp_time</i>	<i>max_comp_time_type</i>	xs:unsignedShort ms
<i>msg_id</i>	<i>msg_id_type</i>	xs:unsignedInt
<i>parameter_flag</i>		xs:Boolean
<i>private_CA_element</i>	<i>private_ca_element_type</i>	xs:hexBinary
<i>private_CA_element_flag</i>		xs:Boolean
<i>protocol_version</i>	<i>protocol_version_type</i>	xs:unsignedByte
<i>super_cas_id</i>	<i>super_cas_id_type</i>	xs:unsignedInt
NOTE: The <i>commercial_event_information</i> has 'abstract' type. This type shall be redefined in the context of each implementation of ACG⇌EIS protocol according to nature of <i>content_event_information</i> for each <i>super_CAS_id</i> .		

11.7 Parameters Substitution

The ACG uses the *AC_parameters* list in *AC_response* message to notify the EIS that it shall substitute the actual value of the parameters before sending the AC to the SCS using the *SCG_provision* message. The *AC_parameter_tag* is used by ACG to authenticate the specific parameter. Table 56 defines the *AC_parameters_tags* that are used in the ACG⇌EIS protocol. The list can be extended by user-defined values on the basis of an explicit agreement between the ACG and EIS vendors.

Table 56: AC_parameter_tag Value

Parameter name	<i>AC_parameter_tag</i>
<i>start_time</i>	0x6001
<i>event_ID</i>	0x6002
<i>original_network_ID</i>	0x6003
<i>transport_stream_ID</i>	0x6004
<i>service_ID</i>	0x6005
<i>component_ID</i>	0x6006

For each *AC_parameter_tag* from the *AC_response* message, the EIS has to generate a structure in TLV format. The structure shall be as follows:

```
{
    parameter_tag      2 byte
    parameter_length    2 byte
    parameter_value <parameter_length> byte
}
```

Table 57 describes the parameters used by the EIS to create parameter list for EIS⇌SCS protocol. The list can be extended by user-defined values according to an explicit agreement between the ACG and EIS vendors.

Table 57: AC_parameter TLV protocol

Parameter Name	Parameter Tag Value	Units	Length (Bytes)
<i>start_time</i>	0x6001	UTCMJD	5
<i>event_ID</i>	0x6002	<i>uimsbf</i>	2
<i>original_network_ID</i>	0x6003	<i>uimsbf</i>	2
<i>transport_stream_ID</i>	0x6004	<i>uimsbf</i>	2
<i>service_ID</i>	0x6005	<i>uimsbf</i>	2
<i>component_ID</i>	0x6006	<i>uimsbf</i>	2

Within the *SCG_provision* message in the EIS⇌SCS protocol, the EIS shall send as an access criteria field the following structure, where *ACG_access_criteria* is the *AC_blob* it gets from the ACG, and where the parameter value loop contains the different parameter it may be asked to add by the ACG. And if no parameter is requested this loop is left empty.

```

access_criteria()
{
    access_criteria_tag      2 byte
    access_criteria_length   2 byte
    {
        ACG_access_criteria
        for(i=0;i<n;i++)
        {
            AC_parameter_tag      2 byte
            AC_parameter_length    2 byte
            AC_parameter_value    <AC_param_length> byte
        }
    }
}

```

Where *n* is an *AC_parameter* identifier.

The value of *access_criteria_tag* is 0x000D.

If the ACG requests from the EIS the value of *component_ID* or *service_ID* and SCG of the event contains more than one instance of *component_ID* or *service_ID*, then all values of *component_ID* or *service_ID* shall be inserted in parameter list and *n* shall be increased appropriately.

11.8 Channel Specific Messages

Each message denotes the direction the message may be passed:

- EIS⇒ACG: indicates EIS may send this message to ACG only;
- EIS⇌ACG: indicates ACG may send this message to EIS only;
- EIS⇌ACG: indicates this message may be sent by either EIS or ACG.

11.8.1 The *channel_setup* Message (EIS⇒ACG)

The *channel_setup* message is sent from the EIS to the ACG when the connection is first opened. Once a channel has been established, this message shall not be sent. This message also dictates certain aspects of the EIS behaviour for this interface.

The SCS shall respond with a *channel_status* message or with a *channel_error* message in the case where the *ACG_channel_ID* is already in use.

Table 58: The *channel_setup* Element

Diagram	channel_setup			
Type	channel_setup_message_type			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	MSG_ID	MSG_ID_type	Required	
	super_CAS_ID	super_CAS_ID_type	Required	
	parameter_flag	xs:boolean	Required	
	interrupt_flag	xs:boolean	Required	
Source	<xs:element name="channel_setup" type="channel_setup_message_type"/>			

11.8.2 The *channel_test* Message (EIS↔ACG)

A *channel_test* message may be sent by either the EIS or the ACG to check if the connection is still alive and error-free.

The peer shall respond with a *channel_status* message if the channel is free of errors, or a *channel_error* message. If the peer does not respond within a reasonable amount of time, the sender may assume the connection is no longer valid and it may close the connection.

Table 59: The *channel_test* Element

Diagram	channel_test			
Type	channel_test_message_type			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	msg_ID	xs:unsignedShort	Required	
Source	<xs:element name="channel_test" type="channel_test_message_type"/>			

11.8.3 The *channel_status* Message (EIS↔ACG)

The *channel_status* message (*message_type* = 0x0403) is sent in response to a *channel_setup* message or *channel_test* message. This message defines the channel parameters between the EIS and the ACG, as well as the ACG behaviour in this protocol as described in clause 11.5.

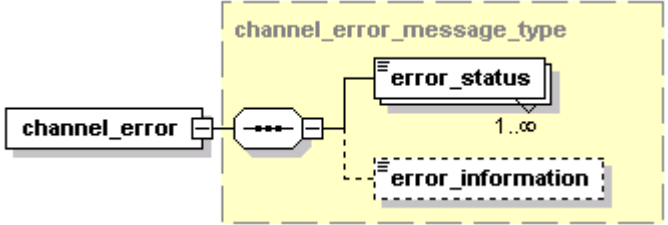
Table 60: The *channel_status* Element

Diagram	channel_status			
Type	channel_status_message_type			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	MSG_ID	MSG_ID_type	Required	
	Max_comp_time	max_comp_time_type	Required	
	private_CA_element_flag	xs:boolean	Required	
	interrupt_flag	xs:boolean	Required	
	parameters_flag	xs:boolean	Required	
Source	<xs:element name="channel_status" type="channel_status_message_type"/>			

11.8.4 The *channel_error* Message (EIS⇌ACG)

A *channel_error* message is sent by the recipient of a *channel_test* message or by the ACG at any time to indicate that an unrecoverable channel level error occurred. A table with possible error conditions appears in clause 11.10.


Table 61: The *channel_error* Element

Diagram				
Type	channel_error_message_type			
Children	error_status error_information			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	msg_ID	xs:unsignedShort	Required	
Source	<xs:element name="channel_error" type="channel_error_message_type" maxOccurs="unbounded"/>			

11.8.5 The *channel_close* Message (EIS⇒ACG)

A *channel_close* message shall be sent by the EIS to indicate the channel is to be closed. No more messages shall be exchanged between the ACG and the EIS.

Table 62: The *channel_close* Element

Diagram				
Type	channel_close_message_type			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	msg_ID	xs:unsignedShort	Required	
Source	<xs:element name="channel_close" type="channel_close_message_type"/>			

11.9 Messages of Access Criteria Creation and Modification

11.9.1 The *AC_request* Message (EIS⇒ACG)

The *AC_request* message is sent from the EIS to ACG to provide the ACG with the *commercial_event_information* parameter that will result in the generation of the *ACG_access_criteria_struct(s)*. See clause 11.3.11. The *commercial_event_information* parameter can be different for each *super_CAS_id*.

Table 63: The *AC_request* Element



Diagram				
Type	AC_request_message_type			
Children	commercial_event_info			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	msg_ID	xs:unsignedShort	Required	
Source	<xs:element name="AC_request" type="AC_request_message_type" maxOccurs="unbounded"/>			

Table 64: The *commercial_event_info* Element

Diagram	
Type	commercial_event_info_type
Source	<code><xs:element name="commercial_event_info" type="commercial_event_info_type"/></code>

11.9.2 The *AC_response* Message (EIS⇔ACG)

The *AC_response_message* is a reply to the *AC_request* message. It carries the *access_criteria_struct*(s), computed by the ACG from the information provided by the *AC_request* message (and possibly from other CA specific information).

The *AC_response_message* can contain several *access_criteria_struct*(s) as defined in clause 11.3.9. All parameters of the *access_criteria_struct* are optional, except for the *ACG_access_criteria*.

The *AC_parameter*(s) can be used only in the case when both the EIS and the ACG support the parameter mode. This should be signalled both in the *channel_setup* and the *channel_status* messages by using the *parameter_flag*. See clause 11.3.12 for a detailed explanation.

The *AC_id* parameter can be used only in the case when both the EIS and the ACG support asynchronous interrupt mode. It should be signalled both in the *channel_setup* and the *channel_status* messages by using *interrupt_flag*. See clause 11.3.11 for a detailed explanation.

The *AC_expiration_time* parameter can be used to signal the EIS that the appropriate *ACG_access_criteria* has to be updated after this time interval will expire. See clause 11.3.10 for a detailed explanation.

The *private_CA_element* parameter can be used only in the case when the *private_CA_element_flag* in the *channel_status* message has the value "TRUE". See clause 11.5 for a further explanation.

Table 65: The *AC_response* Element


Diagram				
Type	AC_response_message_type			
Children	access_criteria_structure			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	MSG_ID	MSG_ID_type	Required	
Source	<code><xs:element name="AC_response" type="AC_response_message_type" minOccurs="0" maxOccurs="unbounded"/></code>			

Table 66: The *access_criteria_structure* Element

Diagram				
Type	<u>access_criteria_structure_type</u>			
Children	<u>ACG_access_criteria</u> <u>AC_parameter</u> <u>private_CA_element</u>			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	AC_ID	AC_ID_type	Optional	
	AC_expiration_time	AC_expiration_time_type	Optional	
Source	<xs:element name="access_criteria_structure" type="access_criteria_structure_type" maxOccurs="unbounded"/>			

11.9.3 The *AC_interrupt* Message (EIS⇌ACG)

The *AC_interrupt* message is sent from the ACG to the EIS in order to signal that the *ACG_access_criteria* corresponds to the *AC_id* must be changed to the new *ACG_access_criteria* value present in this message. See clause 11.3.11 for a detailed explanation.

The use of *AC_parameter*, *private_CA_element*, and *AC_expiration_time* is the same as for the *AC_response* message. See clause 11.9.2.

Table 67: The *AC_interrupt* Element

Diagram				
Type	<u>AC_interrupt_message_type</u>			
Children	<u>ACG_access_criteria</u> <u>AC_parameter</u> <u>private_CA_element</u>			
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	MSG_ID	MSG_ID_type	Required	
	AC_ID	AC_ID_type	Required	
	AC_expiration_time	AC_expiration_time_type	Optional	
Source	<xs:element name="AC_interrupt" type="AC_interrupt_message_type" maxOccurs="unbounded"/>			

11.9.4 The *AC_error* Message (EIS⇌ACG)

An *AC_error* message is sent by the ACG to EIS as a response to an *AC_request* message to indicate that an error in AC request is occurred. A table with possible error conditions appears in clause 11.10.

Table 68: The AC_error Element

Diagram									
Type	<u>AC_error_message_type</u>								
Children	<u>error_status</u> <u>error_information</u>								
Attributes	<table><tr><th><u>Name</u></th><th><u>Type</u></th><th><u>Use</u></th><th><u>Default</u></th></tr><tr><td>msg_ID</td><td>xs:unsignedShort</td><td>Required</td><td></td></tr></table>	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>	msg_ID	xs:unsignedShort	Required	
<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>						
msg_ID	xs:unsignedShort	Required							
Source	<xs:element name="AC_error" type="AC_error_message_type" minOccurs="0" maxOccurs="unbounded"/>								

11.10 Error Status

It is up to the ACG to determine the most appropriate step to be taken for each error. However, an 'unrecoverable error' explicitly means that the channel has to be closed. Most of the errors listed below cannot occur in normal operation. They are mainly provided to facilitate the integration and debugging phase.

Table 69 describes the error statuses.

Table 69: Protocol Error Status Value

Error Status Value	Error Type
0x0000	DVB Reserved
0x0001	Invalid message
0x0002	Unsupported protocol version
0x0003	Message too long
0x0004	Inconsistent type of parameter
0x0005	Missing mandatory parameter
0x0006	Invalid value of parameter
0x0007	Unknown ACG_channel_ID value
0x0008	Unknown AC_ID value
0x0009	Interrupt mode not supported
0x000A	Parameter mode not supported
0x000B	Unknown parameter tag
0x000C to 0x6FFF	Reserved
0x7000	Unknown error
0x7001	Unrecoverable error
0x7002 to 0x7FFF	Reserved
0x8000 to 0xFFFF	EIS/CAS specific/User defined

NOTE: TCP connection level errors are beyond the scope of the present document.

12 SIMCOMP ⇔ MUXCONFIG Interface

12.1 Overview

The present document presents an interface between the MUXCONFIG component and any other Simulcrypt component (SIMCOMP) wishing to transmit data to a MUX for insertion in the Transport Stream generated by that MUX. The purpose of the present document is to define the required interface for allowing various SIMCOMPs to interact with the MUXCONFIG for the purpose of multiplexer redundancy management. This interface is mainly targeted at the EMMG, PSIG, SIG, SCS and EIS components.

The MUXCONFIG is any software application or module that drives head-end compression equipment, most notably the multiplexer.

In a typical Simulcrypt head-end there are several SIMCOMPs that are connected to a multiplexer. This typically includes the EMMG, PSIG, SCS, and the MUXCONFIG. The configuration, provisioning, and redundancy management of a multiplexer is done internally by the MUXCONFIG. The SIMCOMPs mainly connect to the multiplexer to feed data into it.

Multiplexers managed by MUXCONFIG typically allow for 1:1 hot redundancy, or N:M cold redundancy management through a pool of spare multiplexers. In a 1:1 redundancy scenario it may still be possible to statically determine the IP address of the backup multiplexer, but in N:M scenarios it is impossible to know which device is going to be the backup until the real switch-over takes place (or is about to take place).

This interface focuses primarily on two main functions:

- The first is to provide the physical properties of the multiplexer to any SIMCOMP that wishes to connect to the multiplexer for a specific transport and a specific protocol.
- The second is to notify the SIMCOMP of any physical device-to-transport mapping changes. This is needed primarily to facilitate various redundancy scenarios (including 1:1 and N:M) that result in the transport moving to a different physical device within the MUXCONFIG. The SIMCOMP can then use the IP properties of the new multiplexer and connect to it.

This interface does not deal with the redundancy management of higher-level SIMCOMPs, such as the ECMG, EMMG, PSIG, EIS, or that of MUXCONFIG itself. This proposal is only targeted towards notifying these components about a target multiplexer for a specified transport (identified by *transport_stream_id* and *original_network_id*).

This protocol is not intended to replace the SIMF protocol as described in clause 7. While some of the same information is available in both protocols, they are accessed in fundamentally different ways. When both protocols are implemented on the same system, the information common to them should be consistent.

12.2 Interface Principles

12.2.1 System Diagram

The diagram below shows the interactions between the MUXCONFIG, and various other Simulcrypt components like the EIS, the PSIG and the EMMG. The SIMCOMP and the MUXCONFIG communicate via the TCP/IP connection.

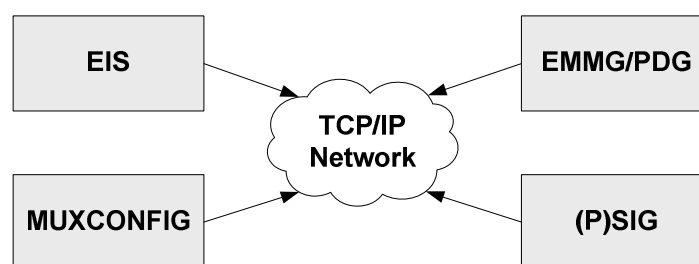


Figure 25: System Diagram for SIMCOMP ↔ MUXCONFIG

In the communication between the SIMCOMP and MUXCONFIG, the MUXCONFIG is a TCP server listening on a well-known socket port and SIMCOMP is a client, which initiates the TCP connection to the server.

There can be several MUXCONFIGs within the network, each controlling a subset of the transport space in the system.

The SIMCOMP has prior knowledge of which MUXCONFIGs are present in the system and may discover which Transport Streams are managed by each MUXCONFIG.

In response to the request message, the MUXCONFIG will return the transport mappings only for the specified transports. In cases of redundancy switch-over initiated/detected by MUXCONFIG, it will send out update message for all the transports for which the physical multiplexer has changed.

12.2.2 Data Communications and Message Format

All messages exchanged between the SIMCOMP and the MUXCONFIG share a common general format as detailed in clause 4.4.2. The messages are encoded using XML over a TCP connection. The XML Schema Definition for this protocol is described in annex L.

Each *message_data* section: begins with a <message> tag. Under each message tag there is a root node for each message and is the same as the name of the message. The root node has just the *channel_id* and the node name is used to identify the message. This is shown in the following example:

```
<message>
  <channel_test channel_id="1"/>
</message>
```

12.2.3 Message Groups

The SIMCOMP⇔MUXCONFIG interface has two different message sub-groups.

12.2.3.1 Communication channel messages

This group of messages help in setting up and managing a TCP communication channel between SIMCOMP and MUXCONFIG. It also allows for polling of the communication status between the two types of components.

Table 70: Communication Channel Messages

ID	Message name	Interface	Purpose
1	<i>channel_setup</i>	SIMCOMP ⇔ MUXCONFIG	The SIMCOMP sends this message after establishing a TCP connection with the MUXCONFIG on a predefined and agreed upon port.
2	<i>channel_status</i>	SIMCOMP ⇔ MUXCONFIG	Initial Response to SIMCOMP on the established connection. If the MUXCONFIG accepts the connection, the return code is 0, otherwise appropriate return code is returned in <i>channel_error</i> message. This message is sent at any time in response to the <i>channel_test</i> message and could be sent either by MUXCONFIG or SIMCOMP.
3	<i>channel_test</i>	SIMCOMP ⇔ MUXCONFIG	Typically SIMCOMP sends a test message to check the server channel status. The response to this message is the <i>channel_status</i> message. This message may also be sent at any time by MUXCONFIG to verify the connection status at the SIMCOMP. The other party returns the <i>channel_status</i> or <i>channel_error</i> message in response to this request.
4	<i>channel_close</i>	SIMCOMP ⇔ MUXCONFIG	SIMCOMP send this message to the MUXCONFIG, if it wishes to close the channel, so that any required cleanup may be done. The channel can also be closed by MUXCONFIG directly and corresponding message in this case is sent to the SIMCOMP connected on that channel.
5	<i>channel_error</i>	SIMCOMP ⇔ MUXCONFIG	MUXCONFIG send this message to the SIMCOMP, whenever it detects an unrecoverable error in the channel communication, or in response to a <i>channel_test</i> or <i>channel_setup</i> message.

12.2.3.2 Transport resource mapping messages

This group of interface messages enables any SIMCOMP component to obtain the physical device to transport mapping and get event updates for transports that move on to a different physical device within MUXCONFIG. The physical properties of the multiplexer can be different for each SIMCOMP or protocol type.

Table 71: Transport Resource Mapping Messages

ID	Message name	Interface	Purpose
6	<i>TS_resource_discover</i>	SIMCOMP ⇔ MUXCONFIG	The SIMCOMP sends this message to MUXCONFIG to discover the mapping between all transports that exist in that MUXCONFIG and the IP properties (per protocol) of the related physical devices. The response to this message shall be a <i>TS_resource_update</i> message.
7	<i>TS_resource_request</i>	SIMCOMP ⇔ MUXCONFIG	The SIMCOMP sends this message to MUXCONFIG to request the mapping between one or more transports and the IP properties (per protocol) of the related physical devices. The protocol of interest must be specified for each transport. The response to this message shall be a <i>TS_resource_update</i> message.
8	<i>TS_resource_update</i>	SIMCOMP ⇔ MUXCONFIG	The MUXCONFIG responds with this message for every <i>TS_resource_request</i> or <i>TS_resource_discover</i> and returns a requested list of transport to <i>IP_address</i> and socket port mapping per requested protocol. In response to the <i>TS_resource_discover</i> message, the data for all the transports that the MUXCONFIG manages is returned. MUXCONFIG asynchronously sends this message at the time of multiplexer redundancy.

12.2.4 Channel Establishment

For the SIMCOMP and the MUXCONFIG to communicate, SIMCOMP will open a TCP/IP connection on a pre-defined socket, send a request and receive a response. The MUXCONFIG can also send asynchronous messages to SIMCOMP for redundancy replacement and channel test.

The initial communication begins with the MUXCONFIG listening on a socket and the SIMCOMP opening a TCP Connection to the MUXCONFIG via that socket. The SIMCOMP sends a *channel_setup* message to the MUXCONFIG. This socket connection is used for all further communication between the MUXCONFIG and the SIMCOMP. **Both the client and the server should use an agreed upon socket port for communications. The recommended TCP port used for initial communication is 57001.**

There is always one-and-only-one channel per TCP connection. Once the TCP connection is established, the SIMCOMP sends a *channel_setup* message to the MUXCONFIG. In the case of success, the MUXCONFIG replies with a *channel_status* message.

In the case of a rejection or failure during channel setup the MUXCONFIG replies with a *channel_error* message. This means that the channel is not established and the SIMCOMP shall close the TCP connection. In case of *channel_error* the MUXCONFIG should also close the channel on its side.

12.2.5 Timeout and Retry

The SIMCOMP shall be able to handle an unexpected communication loss (either on the connection, channel or message level). Each component, when suspecting a possible communication loss should check the communication status by sending a test message and expect to receive a status message. If the status message is not received in the given time (implementation specific) the communication path should be re-established or a timeout explicitly flagged to the user of the system.

12.2.6 Channel closure

Channel closure can occur when the channel is no longer needed or in case of error. This is done by means of a *channel_close* message sent by either the SIMCOMP or the MUXCONFIG. Subsequently, both sides shall close the connection.

If the channel is closed or if the connection is broken, current associations ((*transport_id*, *original_network_id*, *mux_protocol_type*), (*ip_address*, *port_number*)) may be used by SIMCOMP until the connection is re-established. Once the connection is re-established, the SIMCOMP shall request all the resource resolution information it needs and update its connections as needed.

12.2.7 Channel testing and status

After a channel has been established, either the SIMCOMP or the MUXCONFIG may send a *channel_test* message to check the integrity of a channel. In response to this message the receiving component shall reply with a *channel_status* message or *channel_error* message.

12.2.8 Handling data inconsistencies

If the MUXCONFIG or SIMCOMP detects an inconsistency it shall send an error message to the other party. After receiving the error message, the SIMCOMP may close the connection. The SIMCOMP (as the client) may then re-establish the connection.

12.3 Parameter Semantics

channel_id: This parameter shall uniquely identify the TCP connection between the SIMCOMP and the MUXCONFIG. The *channel_id* is a numerical identifier for the channel. The *channel_id* is a unique number and must be agreed for each SIMCOMP so that any overlap or duplication is avoided.

client_id: This parameter shall uniquely identify an EMMG/PDG across all the EMMGs/PDGs connected to a given mux. In order to facilitate uniqueness of this value, the following rules apply:

- In the case of EMMs or other CA related data, the two first bytes of the *client_id* should be equal to the two bytes of the corresponding *CA_system_id*. In other cases a value allocated by DVB for this purpose should be used.

protocol_version: This indicates the version of the SIMCOMP⇌MUXCONFIG protocol in use. It is sent at connection establishment time to ensure compatible revisions are being used. **The value of *protocol_version* shall be 0x04.**

mux_protocol_type: This parameter indicates which kind of protocol is going to be used by the SIMCOMP to communicate with the requested MUX. The possible values allowed are shown in table 72.

Table 72: Possible values for the *mux_protocol_type* parameter

Value	Target Protocol	Type of Data
0x00	All Protocols	Used to request information on all protocols in use
0x01	EMMG ⇌ MUX TCP	EMM
0x02	EMMG ⇌ MUX UDP	EMM
0x03	PDG ⇌ MUX TCP	Private Data
0x04	PDG ⇌ MUX UDP	Private Data
0x05	PSIG ⇌ MUX	PSI Tables
0x06	SIG ⇌ MUX	SI Tables
0x07	EIS ⇌ SCS	Scrambling group Provisioning
0x08	SCS ⇌ MUX (see note)	ECM + CW
0x09 to 0x7F	Reserved	Reserved for future use
0x80 to 0xFF	User Defined	Private user defined data
NOTE: This is not a standardized protocol.		

ip_address: This parameter shall indicate the IP address that the SIMCOMP must use to connect (using the protocol identified by the *mux_protocol_type* parameter) to the MUX currently generating the requested transport stream that is identified by the (*original_network_id*, *transport_stream_id*) parameters.

port_number: This parameter shall indicate the TCP port that the SIMCOMP must use to connect to the MUX that is currently generating the requested transport stream identified by the (*original_network_id*, *transport_stream_id*) parameters. The protocol that should be used by the SIMCOMP on this connection is identified by the *mux_protocol_type* attribute.

original_network_id: This parameter with the *transport_stream_id* parameter shall uniquely identify the MPEG transport stream within the network. Refer to the DVB SI specification (EN 300 468 [1]) and MPEG-2 system (ISO/IEC 13818-1 [3]) for more information.

transport_stream_id: This parameter associated to the *original_network_id* parameter shall uniquely identify the MPEG transport stream within the network. Refer to the DVB SI specification (EN 300 468 [1]) and MPEG-2 system (ISO/IEC 13818-1 [3]) for more information.

is_allocated: This parameter indicates whether each mapping of the transport with the device is being currently allocated or de-allocated. This is a Boolean value and TRUE means that the mapping is currently allocated.

description: This parameter provides human-readable information relating to a particular response.

response_code: This code is returned in response messages to indicate the status of a request, according to table 73.

Table 73: Response codes

Response Code	Type	Description
0x0000	Normal Response	Indicates a successful response.
0x0001	Invalid XML message	Message size does not correspond with the size of the data read or the XML failed validation
0x0002	Unsupported protocol version	Unsupported protocol version
0x0003	Unknown message type	The format of the XML expected for the message is different from that obtained or the message type is not supported
0x0004	Duplicate channel id	A <i>channel_id</i> that is already in use was passed again for a new connection.
0x0005	Unsupported mux protocol type	The requested mux protocol type is not supported by MUXCONFIG.
0x0006	Unknown transport	The specified transport is unknown to MUXCONFIG.
0x0007 to 0x6FFF	DVB Reserved	
0x7000	Unknown error	Unknown failure occurred.
0x7001	Unrecoverable error	Unrecoverable failure occurred
0x7002 to 0x7FFF	DVB Reserved	
0x8000 to 0xFFFF	User Definable	

12.4 Interface Structure

The root element for all messages on the SIMCOMP ⇌ MUXCONFIG interface is the *message* element. Each message element may contain one message specific child element. The message specific child elements are described in clauses 12.5 and 12.6. The XML Schema Definition for this protocol is described in annex L.

Table 74: The *message* Element

Diagram	
Children	<u>channel_setup</u> <u>channel_status</u> <u>channel_test</u> <u>channel_close</u> <u>channel_error</u> <u>TS_resource_request</u> <u>TS_resource_update</u> <u>TS_resource_discover</u>
Source	<pre> <xs:element name="message"> <xs:complexType> <xs:choice> <xs:element ref="channel_setup" minOccurs="0"/> <xs:element ref="channel_status" minOccurs="0"/> <xs:element ref="channel_test" minOccurs="0"/> <xs:element ref="channel_close" minOccurs="0"/> <xs:element ref="channel_error" minOccurs="0"/> <xs:element ref="TS_resource_request" minOccurs="0"/> <xs:element ref="TS_resource_update" minOccurs="0"/> <xs:element ref="TS_resource_discover" minOccurs="0"/> </xs:choice> </xs:complexType> </xs:element> </pre>


12.5 Channel Specific Messages

12.5.1 The *channel_setup* message (SIMCOMP ⇌ MUXCONFIG)

The *channel_setup* message is sent by the SIMCOMP to the MUXCONFIG in order to set up a channel once the TCP connection has been established. It contains the *channel_id* and the *protocol_version*. The SIMCOMP *channel_id* will uniquely identify this connection in the MUXCONFIG.

The SIMCOMP allocates the *channel_id*. If the SIMCOMP attempts to make a connection to the MUXCONFIG by specifying a *channel_id* that is already in use, the *channel_status* message received in response to the *channel_setup* request will contain a response code of 0x0004 (duplicate channel ID), indicating that the specified *channel_id* is already in use.

Table 75: The *channel_setup* element

Diagram				
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
	protocol_version	xs:unsignedByte	required	
Source	<pre> <xs:element name="channel_setup"> <xs:complexType> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> <xs:attribute name="protocol_version" type="xs:unsignedByte" use="required"/> </xs:complexType> </xs:element> </pre>			

12.5.2 The *channel_status* message (SIMCOMP ⇔ MUXCONFIG)

The *channel_status* message is a reply to the *channel_setup* or *channel_test* message. It indicates that the *channel_setup* or *channel_test* message was received and validated. This message contains a response code and optional description parameter that specify any errors or specific reasons for failure.

Optionally this message also contains the node to indicate what protocols are supported and will be returned only in the message response to *channel_setup* message.

Table 76: The *channel_status* element

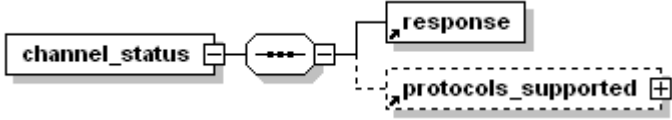
Diagram				
Children	<u>response protocols_supported</u>			
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
Source	<pre> <xs:element name="channel_status"> <xs:complexType> <xs:sequence> <xs:element ref="response"/> <xs:element ref="protocols_supported" minOccurs="0"/> </xs:sequence> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element> </pre>			

Table 77: The *response* element


Diagram				
Used by	elements	<u>channel_error channel_status TS_resource_update</u>		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	response_code	xs:unsignedShort	required	
	description	xs:string	optional	
Source	<pre> <xs:element name="response"> <xs:complexType> <xs:attribute name="response_code" type="xs:unsignedShort" use="required"/> <xs:attribute name="description" type="xs:string" use="optional"/> </xs:complexType> </xs:element> </pre>			

Table 78: The *protocols_supported* element

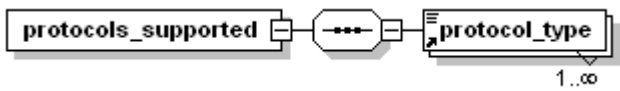

Diagram	
Children	<u>protocol_type</u>
Used by	element <u>channel_status</u>
Source	<pre> <xs:element name="protocols_supported"> <xs:complexType> <xs:sequence> <xs:element ref="protocol_type" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element> </pre>

Table 79: The *protocol_type* element

Diagram	
Type	xs:unsignedByte
Used by	element <u>protocols_supported</u>
Source	<pre> <xs:element name="protocol_type" type="xs:unsignedByte"/> </pre>

12.5.3 The *channel_test* message (SIMCOMP ⇔ MUXCONFIG)

The *channel_test* message is sent by either the SIMCOMP or MUXCONFIG to test the connection and also serves as a polling function, which the SIMCOMP can use to monitor the status of the connection.

This message does not contain any parameters.


Table 80: The *channel_test* element

Diagram	<div><div>channel_test</div></div>								
Used by	element <u>message</u>								
Attributes	<table><tr><th><u>Name</u></th><th><u>Type</u></th><th><u>Use</u></th><th><u>Default</u></th></tr><tr><td>channel_id</td><td>xs:unsignedShort</td><td>required</td><td></td></tr></table>	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>	channel_id	xs:unsignedShort	required	
<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>						
channel_id	xs:unsignedShort	required							
Source	<xs:element name="channel_test"> <xs:complexType> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element>								

12.5.4 The *channel_close* message (SIMCOMP ⇔ MUXCONFIG)

The *channel_close* message is sent by the SIMCOMP to indicate that the channel is to be closed. MUXCONFIG will free any associated resources and no further updates for the transports for this channel will be generated.

Table 81: The *channel_test* element

Diagram				
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
Source	<pre><xs:element name="channel_close"> <xs:complexType> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element></pre>			

12.5.5 The *channel_error* message (SIMCOMP ⇐ MUXCONFIG)

The *channel_error* message is an asynchronous message from MUXCONFIG. It indicates that an unrecoverable error has been detected in the channel communication and that the channel is no longer valid. This message can contain a *response_code* if appropriate. See the table for appropriate error codes.

This message is sent from the MUXCONFIG in response to a *channel_setup* or a *channel_test* message in the event of failure. The *description* parameter specifies any additional errors description or the specific reason for failure.

Table 82: The *channel_error* element

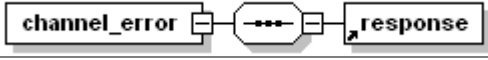
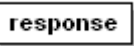
Diagram				
Children	response			
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
Source	<pre><xs:element name="channel_error"> <xs:complexType> <xs:sequence> <xs:element ref="response"/> </xs:sequence> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element></pre>			

Table 83: The *response* element

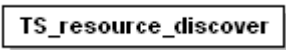
Diagram				
Used by	elements	channel_error channel_status TS_resource_update		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	response_code	xs:unsignedShort	required	
	description	xs:string	optional	
Source	<pre><xs:element name="response"> <xs:complexType> <xs:attribute name="response_code" type="xs:unsignedShort" use="required"/> <xs:attribute name="description" type="xs:string" use="optional"/> </xs:complexType> </xs:element></pre>			

12.6 Transport Resource Mapping Messages

12.6.1 The *TS_resource_discover* message (SIMCOMP ⇌ MUXCONFIG)

The SIMCOMP sends this message to MUXCONFIG to obtain the physical to logical mapping for all the transports and protocols that the MUXCONFIG is currently managing. This should be the first message after successful connection setup that the SIMCOMP will send to the MUXCONFIG to obtain the current mapping of the transport to the physical devices. The MUXCONFIG shall send *TS_resource_update* message in response to this message.

Table 84: The *TS_resource_discover* element

Diagram				
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
Source	<pre> <xs:element name="TS_resource_discover"> <xs:complexType> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element> </pre>			

12.6.2 The *TS_resource_request* message (SIMCOMP ⇌ MUXCONFIG)

The SIMCOMP will send this message to MUXCONFIG to obtain the physical to logical mapping for a specified transport. The message must contain the *original_network_id*, *transport_stream_id*, and *mux_protocol_type* parameters, which collectively form the unique resource identifier. A *client_id* parameter may be used to narrow the request to a particular EMMG/PDG connection when requesting EMMG/PDG protocol information. Multiple resource identifiers may be specified in a single *TS_resource_request* message.

The *mux_protocol_type* is an enumeration of various Simulcrypt protocols such as the EIS⇌SCS, PSIG⇌MUX or PDG⇌MUX protocol to indicate to the MUXCONFIG which protocol the SIMCOMP is interested in for the specified transport. One of the protocol enumeration types refers to all defined protocols. Using this enumeration value indicates to the MUXCONFIG that the SIMCOMP is requesting information for all the supported protocols.

If the *client_id* attribute is missing, then the response to this request is not filtered based on the *client_id* and all information for the associated transport and protocol is returned. If the *client_id* is provided then the MUXCONFIG filters the reply based on the associated *client_id*.

The MUXCONFIG responds with a list of transports that match the request, along with their physical mapping in the corresponding response message.

Table 85: The *TS_resource_request* element

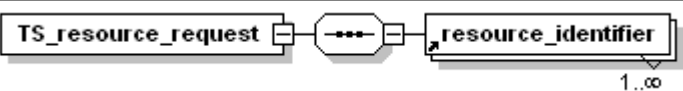
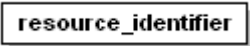
Diagram				
Children	resource_identifier			
Used by	element	message		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	channel_id	xs:unsignedShort	required	
Source	<pre> <xs:element name="TS_resource_request"> <xs:complexType> <xs:sequence> <xs:element ref="resource_identifier" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element> </pre>			

Table 86: The *resource_identifier* element

Diagram				
Used by	element	TS_resource_request		
Attributes	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>
	original_network_id	xs:unsignedShort	required	
	transport_stream_id	xs:unsignedShort	required	
	mux_protocol_type	xs:unsignedByte	required	
	client_id	xs:unsignedInt	optional	
Source	<pre> <xs:element name="resource_identifier"> <xs:complexType> <xs:attribute name="original_network_id" type="xs:unsignedShort" use="required"/> <xs:attribute name="transport_stream_id" type="xs:unsignedShort" use="required"/> <xs:attribute name="mux_protocol_type" type="xs:unsignedByte" use="required"/> <xs:attribute name="client_id" type="xs:unsignedInt" use="optional"/> </xs:complexType> </xs:element> </pre>			

12.6.3 The *TS_resource_update* message (SIMCOMP ⇔ MUXCONFIG)

The MUXCONFIG as part of the *TS_resource_update* message will send the new *IP_address* and the *original_network_id*, *transport_stream_id* mapping information to the SIMCOMP as part of the payload.

The *TS_resource_update* message is sent in response to a *TS_resource_request* message or as an asynchronous update of state changes within the MUXCONFIG domain.

Asynchronous updates are also sent to a SIMCOMP for changes to the physical to logical mapping. Examples of state changes that would cause an asynchronous update include: whenever redundancy switch occurs, whenever a new transport is provisioned or de-allocated within the MUXCONFIG, or whenever the *transport_stream_id* of a transport gets changed within MUXCONFIG. If a *transport_stream_id* changes for a transport, then a message is sent to indicate a de-allocation of the old transport and allocation of a new transport.

A separate *resource_mapping* element is returned for every unique combination of *original_network_id*, *transport_stream_id* and *mux_protocol_type* (and *client_id* if specified in the *TS_resource_request* message). More than one *resource_mapping* element may be returned for each of the above combinations when more than one resource is associated with a given transport/protocol, such as may be the case in 1:1 hot redundancy scenarios.

More than one *resource_mapping* element per transport may also be returned in cases where a single transport is carrying EMMs from more than one CAS vendor or more than one EMMGs.

When used as an asynchronous MUX redundancy replacement message, this message will contain *resource_mapping* elements for only those transports where the status has changed.

In cases where EMMG is using a UDP broadcast connection, but the initial channel handshake is done using a TCP connection, MUXCONFIG will update two *resource_mapping* elements; one for the UDP broadcast port and another for TCP handshake port. The corresponding request in such scenario would also contain two separate requests each with a different protocol type.

If the transport is being de-allocated or deprovisioned within the MUXCONFIG, then there will be a specific *is_allocated* attribute, which will be set to FALSE to indicate that the particular transport identified in the *resource_mapping* element is being deprovisioned. This allows for the EMMG, EIS, or other SIMCOMP to clean up the connection and any data structures that it may have opened on the multiplexer directly. For example, the EIS will cleanup the corresponding SCGs when it receives such a message. Similarly the PSIG will stop the PSI for the transport that has been removed. The attribute *is_allocated* is set to TRUE to indicate that the transport has been allocated and it set to FALSE to indicate that the transport has been de-allocated.

If the *TS_resource_update* message is being sent in response to a *TS_resource_request* message, then the *response* element shall be present; otherwise, it shall not be present. If the MUXCONFIG is unable to respond with a valid *resource_mapping* element for every requested transport stream and protocol type, then the *response* element will indicate that an error has occurred. In such a case, the MUXCONFIG may also populate the *TS_resource_update* message with the *resource_mapping* elements for the requested transport streams and protocol types that are valid.

Table 87: The *TS_resource_update* element

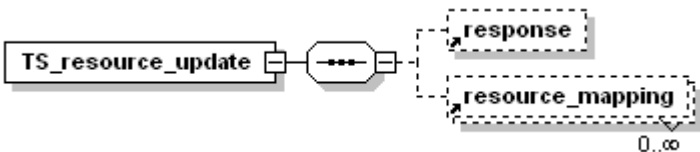
Diagram									
Children	<u>response</u> <u>resource_mapping</u>								
Used by	element <u>message</u>								
Attributes	<table><tr><th><u>Name</u></th><th><u>Type</u></th><th><u>Use</u></th><th><u>Default</u></th></tr><tr><td>channel_id</td><td>xs:unsignedShort</td><td>required</td><td></td></tr></table>	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>	channel_id	xs:unsignedShort	required	
<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>						
channel_id	xs:unsignedShort	required							
Source	<pre><xs:element name="TS_resource_update"> <xs:complexType> <xs:sequence> <xs:element ref="response" minOccurs="0"/> <xs:element ref="resource_mapping" minOccurs="0" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/> </xs:complexType> </xs:element></pre>								

Table 88: The *response* element

Diagram	<div><div>response</div></div>												
Used by	elements <u>channel_error</u> <u>channel_status</u> <u>TS_resource_update</u>												
Attributes	<table><tr><th><u>Name</u></th><th><u>Type</u></th><th><u>Use</u></th><th><u>Default</u></th></tr><tr><td>response_code</td><td>xs:unsignedShort</td><td>required</td><td></td></tr><tr><td>description</td><td>xs:string</td><td>optional</td><td></td></tr></table>	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>	response_code	xs:unsignedShort	required		description	xs:string	optional	
<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>										
response_code	xs:unsignedShort	required											
description	xs:string	optional											
Source	<pre><xs:element name="response"> <xs:complexType> <xs:attribute name="response_code" type="xs:unsignedShort" use="required"/> <xs:attribute name="description" type="xs:string" use="optional"/> </xs:complexType> </xs:element></pre>												

Table 89: The *resource_mapping* element

Diagram	<div><div>resource_mapping</div></div>																																
Used by	element <u>TS_resource_update</u>																																
Attributes	<table><tr><th><u>Name</u></th><th><u>Type</u></th><th><u>Use</u></th><th><u>Default</u></th></tr><tr><td>original_network_id</td><td>xs:unsignedShort</td><td>required</td><td></td></tr><tr><td>transport_stream_id</td><td>xs:unsignedShort</td><td>required</td><td></td></tr><tr><td>mux_protocol_type</td><td>xs:unsignedByte</td><td>required</td><td></td></tr><tr><td>IP_address</td><td>xs:string</td><td>required</td><td></td></tr><tr><td>port_number</td><td>xs:unsignedShort</td><td>required</td><td></td></tr><tr><td>is_allocated</td><td>xs:boolean</td><td>required</td><td></td></tr><tr><td>client_id</td><td>xs:unsignedInt</td><td>optional</td><td></td></tr></table>	<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>	original_network_id	xs:unsignedShort	required		transport_stream_id	xs:unsignedShort	required		mux_protocol_type	xs:unsignedByte	required		IP_address	xs:string	required		port_number	xs:unsignedShort	required		is_allocated	xs:boolean	required		client_id	xs:unsignedInt	optional	
<u>Name</u>	<u>Type</u>	<u>Use</u>	<u>Default</u>																														
original_network_id	xs:unsignedShort	required																															
transport_stream_id	xs:unsignedShort	required																															
mux_protocol_type	xs:unsignedByte	required																															
IP_address	xs:string	required																															
port_number	xs:unsignedShort	required																															
is_allocated	xs:boolean	required																															
client_id	xs:unsignedInt	optional																															
Source	<pre><xs:element name="resource_mapping"> <xs:complexType> <xs:attribute name="original_network_id" type="xs:unsignedShort" use="required"/> <xs:attribute name="transport_stream_id" type="xs:unsignedShort" use="required"/> <xs:attribute name="mux_protocol_type" type="xs:unsignedByte" use="required"/> <xs:attribute name="IP_address" type="xs:string" use="required"/> <xs:attribute name="port_number" type="xs:unsignedShort" use="required"/> <xs:attribute name="is_allocated" type="xs:boolean" use="required"/> <xs:attribute name="client_id" type="xs:unsignedInt" use="optional"/> </xs:complexType> </xs:element></pre>																																

13 Timing and Play-out Issues

13.1 Timing issues

In all systems there is a crypto-period when data is scrambled with a particular Control Word. For STBs to regenerate the CW in time, the ECM play-out has to be correctly synchronized with this CP.

To accommodate different synchronization approaches, the SCS will be responsible for requesting enough CWs and ECM packets in advance of their play-out time. The timing diagram in figure 26 illustrates this relationship between CW generation, ECM generation, ECM play-out and crypto-period.

ECM Timing Diagram

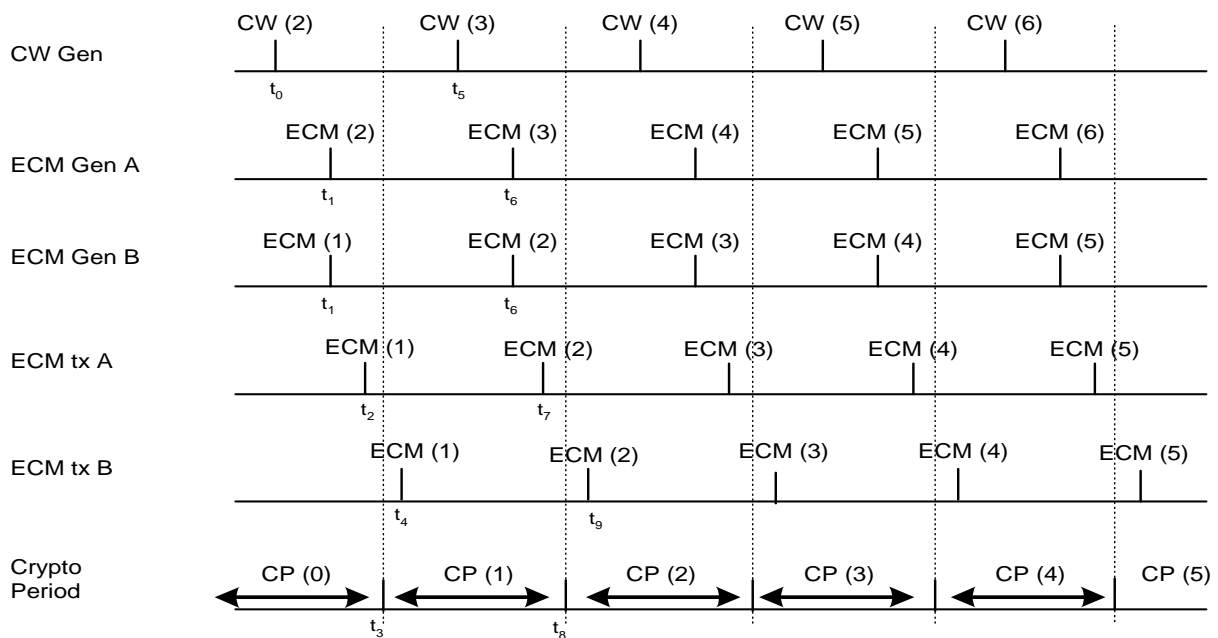


Figure 26: ECM timing diagram

At the beginning of CP(2), at t_8 , the scrambler begins using CW(2) to encrypt the signal. Each CA system shall ensure that its STBs obtain this CW in advance of this point.

CA system A achieves this by producing its ECM for a single CW only. As soon as ECM Gen. A receives CW(2), at t_0 , it is able to produce ECM(2), at t_1 . This ECM(2) is transmitted sufficiently prior to the beginning of CP(2), to ensure that the STB can obtain CW(2) before CP(2).

CA system B achieves the same result by producing its ECM for two CWs. As soon as ECM Gen. B receives CW(2), at t_0 , it is able to produce ECM(1), at t_1 . ECM(1), which encompasses CW(1) and CW(2), is transmitted from the middle of CP(1), at t_4 . This ensures that the STB can obtain CW(2) before CP(2) begins. After CP(2) begins, at t_9 , CW(2) and CW(3) are available in ECM(2).

13.2 Delay Start

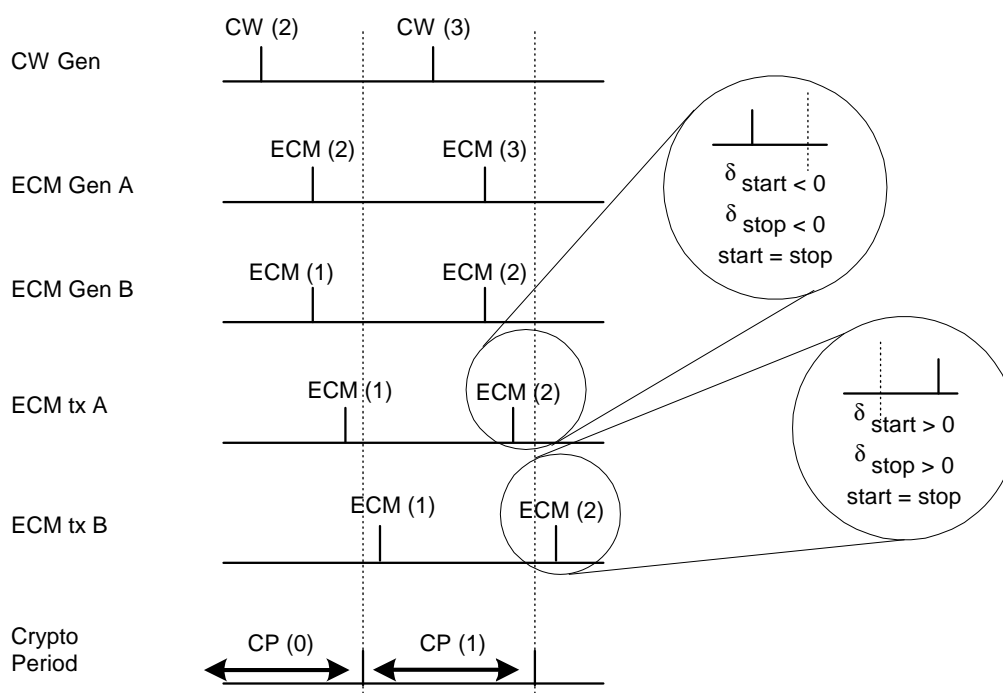


Figure 27: Delay_start and Delay_stop

delay_start: This signed integer represents the amount of time between the start of a crypto-period, and the start of the broadcasting of the ECM attached to this period. If it is positive, it means that the ECM shall be delayed with respect to the start of the crypto-period. If negative, it means that the ECM shall be broadcast ahead of this time. This parameter is communicated by the ECMG to the SCS during the channel setup.

delay_stop: This signed integer represents the amount of time between the end of a crypto-period, and the end of the broadcasting of the ECM attached to this period. If it is positive, it means that the end of the ECM broadcast shall be delayed with respect to the end of the crypto-period. If negative, it means that the ECM broadcast shall be ended ahead of time. This parameter is communicated by the ECMG to the SCS during the channel setup.

It is usual for delay_start and delay_stop to be equal, but this is not mandatory. This is illustrated in figure 28, which shows the case where ECM transmission is stopped *around* a CP boundary.

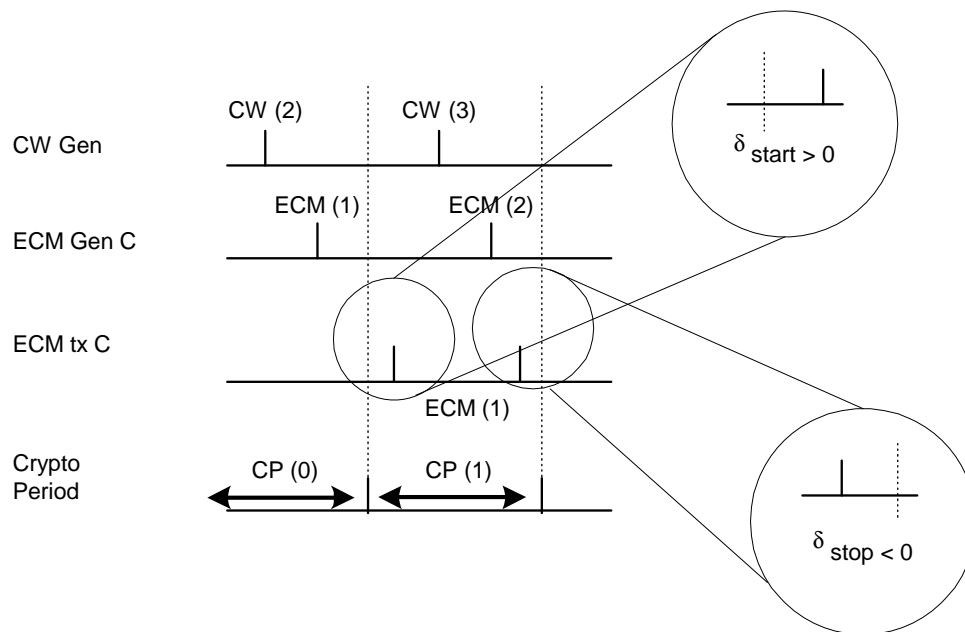


Figure 28: End of ECM transmission around crypto-period boundary

13.3 Play-out Issues

13.3.1 ECMs

When an ECMG sends an ECM_datagram (in the ECM_response message) for a particular ECM stream it implicitly means that:

- the SCS shall trigger the play-out of this ECM at the time calculated with the delay start parameter;
- the SCS shall stop the play-out of the previous ECM of the same stream at the same time;
- in other words the play-out of two ECMs of the same stream can never overlap;
- the play-out of an ECM is also stopped by the SCS when the time calculated with the delay stop parameter is reached.

If an ECMG fails (i.e. the SCS times-out while waiting for an ECM_response message), the SCS has the option to extend the duration of the current crypto-period (e.g. to attempt to reconnect or switch to a backup device). In such a case the play-out of ECMs is extended accordingly.

13.3.2 EMMs and Private Data

The MUX should play-out EMMs/Private Datagrams in the order in which they arrive.

13.4 Crypto-Period Realignment

If a new event starts in the middle of a crypto-period (either from a program or a change in Access Criteria), crypto-periods may need to be re-aligned.

In the case when an activation time is provided in the *SCG_provision* message (as defined in EIS \Leftrightarrow SCS) initiating this realignment, and if the delay between the time when this message is received by the SCS and its associated activation time is longer than the crypto-period, this latter can only be extended. For example, if 21:00:00 starts a new event and the nominal crypto-period duration is 20 seconds, and a crypto-period starts at 20:59:30, then the SCS is not allowed to make a 10 seconds crypto-period between 20:59:50 and 21:00:00 if it received the related *SCG_provision* message more than 20 seconds before *activation_time*. Instead, if it needs to align crypto-periods with the start of the events, it shall lengthen the previous crypto-period to 30 seconds, so that it ends exactly at 21:00:00.

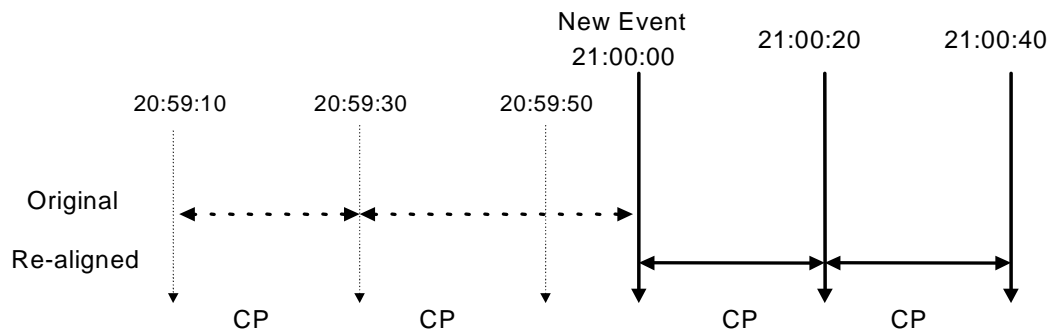


Figure 29: Event realignment

In the special case when the delay is shorter or if no activation time is provided, the SCS is then allowed to shorten the current CP but it is then the SCS's responsibility to make sure that, during this realignment, the crypto-period duration does not drop below:

- all the min_CP_duration specified by the ECMGs during channel_set-up;
- all the max_comp_time values specified by the ECMGs during channel set-up, plus typical network latencies.

Annex A (normative): System Layering

A.1 Introduction

Each clause in this annex describes a single system layer as defined within the OSI model. The presentation layer is not described in the context of the present document.

A.2 Physical Layer

The physical layer provides the physical facilities required to enable the linking together of hosts that need to exchange data.

The physical layer interface shall be ethernet. 10 Base-T (or another fully compatible layer) shall be used on all the interfaces defined by the present document.

A.3 Data Link Layer

The data link layer provides the facilities that allow two hosts that are physically and directly connected (without a third host separating the two) to exchange data. The functionality of the data link layer is covered by the ethernet protocols.

A.4 Network Layer

The network layer provides the facilities to allow two hosts to exchange data directly or indirectly in a network of intervening hosts and gateways.

The network layer, providing point-to-point communication, shall be IP (RFC 791 [9]). Hosts within IP are uniquely identified by their IP address.

A.5 Transport Layer

The transport layer provides the facilities to allow two hosts, either directly or indirectly connected, to exchange data over one or several interconnected networks. Additionally, the transport layer allows communication to take place based on connections between an individually addressable end-point on one host and another individually addressable end-point on the same host or on another host. The transport layer allows as well communication in broadcast mode.

The transport layer shall be TCP (RFC 793 [10]) or UDP (RFC 768 [8]), according to the application protocol.

A.6 Session Layer

The data exchange facility as provided by the session layer to the application layer has the following features:

- Connection Based or Broadcast (*): All communication takes place between two uniquely defined communication end-points, or from one to several communication end-points;
- Sequenced (*): All data transmitted arrives at its destination in the order it was sent;
- Reliable (*): Data integrity is maintained, no data is lost;

- Two-way (*): Both communication end-points of a particular connection can send and receive data;
- Unformatted: The session layer does not impose any structuring on the data it transports; the data presents itself to the receiver as an unformatted data byte stream (data structuring into messages is a responsibility of entities in the application layer).

The features (*) depend on the transport layer protocol (TCP or UDP) or on the session and presentation layers (specific or SNMP).

The number of connections that can be concurrently open at any one time is determined by the operating system under which the applications making use of the stream layer facilities execute. Additionally, in the event of an unexpected connection closure or connection loss and in the event of data read or write errors, the entity in the application layer that opened the connection or performs the read or write operation is notified.

The session layer, providing these facilities, shall be a socket stream interface.

A.7 System Layering Overview/Communications Protocol stack

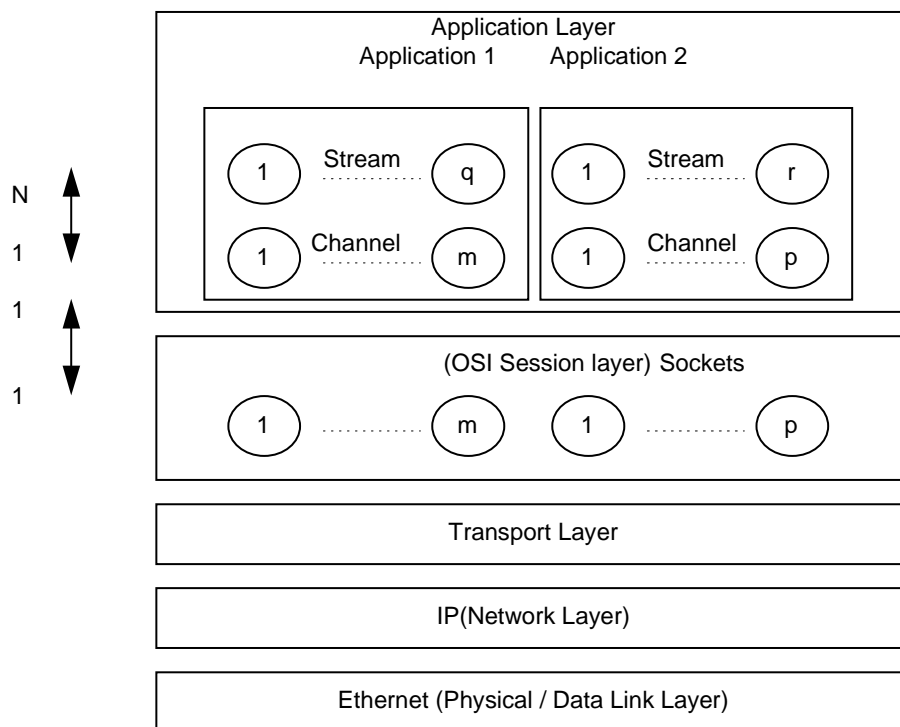


Figure A.1: Systems Layering overview diagram

On the left of this diagram, the mappings between the entities ports, sockets, connections and sessions is depicted:

- "1 <-> 1" indicates a direct association between an instance of an entity of a given type and an instance of a lower layer entity;
- "1 <-> N" indicates that potentially multiple instances of an entity of a given type map onto a single instance of a lower layer entity.

A.8 TCP or UDP Connection Establishment

Connections between client and server are initiated by the client. After establishment of a connection, both client and server have an open socket, identifying the connection, allowing them to exchange data. IP address information required by the client to open a connection is made available by the server in one of two ways:

- Statically: IP address information is defined by static methods;
- Dynamically: This method may use a DNS (Domain Name Server). The DNS can be consulted by the client to retrieve the required information;
- TCP port or UDP port information required by the client to open a connection is made available by the server. Port number information is defined by static methods.

Annex B (informative): SCS Coexistence

B.1 Introduction

This annex describes how the ECMG \Leftrightarrow SCS message interfaces could be used to communicate with another SCS (e.g. in a hot-standby configuration). In a Simulcrypt environment all the information that an SCS needs to communicate with a MUX, is encompassed by the Channel Status, Stream Status, CW Provision and ECM Response messages. This information can be passed to another SCS, which can then use this information to maintain its internal status.

B.2 Example scenario

The EIS will trigger the beginning of a CA event by sending access conditions and start and stop times to SCS₁. SCS₁ will then determine which ECMGs are involved with this CA event. It will establish connections, channels and streams with the appropriate ECMGs. During this establishment, each ECMG will pass, via the Channel and Stream Status messages, all ECMG specific data. SCS₁ will then begin passing CWs to the ECMGs, receiving ECM datagrams in response and synchronize the ECM play-out.

In the example given in figure B.1, SCS₁ will additionally pass all messages received by each ECMG and the information contained in the CW Provision message on to SCS₂. This information can be transmitted to SCS₂ using the same interface as the SCS \Leftrightarrow ECMG. The main difference is that the CW Provision message, which is normally *sent* by SCS₁ to the ECMG, will now be *received* by SCS₂ from SCS₁.

This information will enable SCS₂ to reproduce the environment (connections, channels and streams) running on SCS₁.

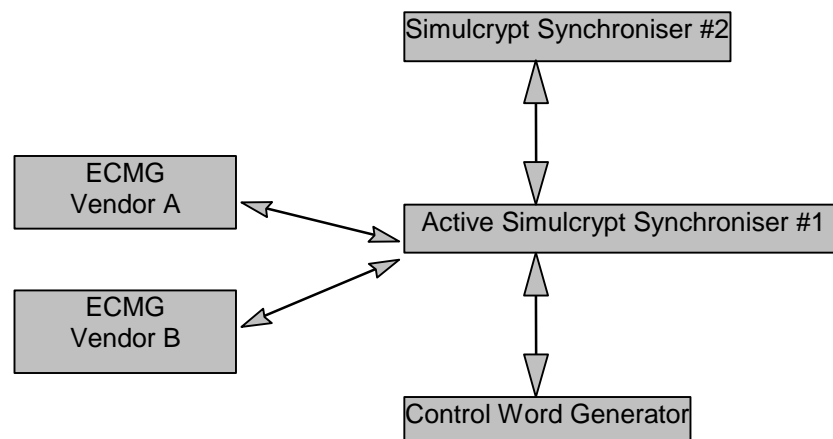


Figure B.1: Example of SCS redundancy

Annex C (informative): Control word generation and testing

C.1 Introduction

The control word generator is an integral part of the DVB Simulcrypt system, which generates low-level cryptographic keys that are used directly to scramble content. It should supply control words that meet certain statistical properties for randomness. Using appropriate (preferably physical) generating techniques and applying statistical tests will reduce to an acceptable level the probability of inadvertently generating control words with deterministic properties.

Since generating highly random control words and applying the appropriate tests for randomness imposes little incremental technical complexity or cost, there is great motivation for implementing the methods described below or their equivalent. Moreover, commercially available hardware and software solutions for this problem make it an easy task to generate control words with high confidence in their randomness qualities.

C.2 Background

The generated control words should approach as nearly as practicable true random sequences. In general, the criteria for producing cryptographically secure random sequences include:

- they have to *appear* random, that is they have to seem to an observer to possess the qualities of true random sequences;
- an observer should not be able to predict the next bit in a sequence even if armed with complete knowledge of the generating algorithm/hardware;
- a given sequence should not be reproducible by running a generator more than once using the same input;

Certifying sequences by applying the statistical tests described below can satisfy criteria 1 and 2, and seeded pseudo random sequences can meet these requirements. However any pseudo random algorithm is just as subject to attack as is an encryption algorithm.

Satisfying Criteria 1 to 3 produces sequences that approach true randomness (by most definitions) and are probably random enough for use as cryptographic keys in most applications, but (3) cannot be achieved using pseudo random techniques. This is not to imply that no pseudo random technique is acceptable for use in generating Simulcrypt control words, only that great care has to be taken to avoid generating sequences that that *appear* random but that can be successfully analysed by an attacker. This is not an easy task, but there are simple tests that can be applied to the algorithm during development that will help insure it satisfies criteria 1 and 2. For instance, an acceptable sequence should not be appreciably compressible (by more than about 1 % or 2 %) using commercial compression programs.

C.3 Generation

The best method to generate random sequences involves using physical phenomena to produce a Gaussian distributed white noise source with a flat magnitude spectrum (± 1 dB, 100 Hz to 120 kHz). Physical methods typically use a thermal or radioactive noise source and are fed to a high-speed comparator to produce a digital output. Such sources are readily available and are simple to construct, and their output cannot be replicated even though an attacker may possess an exact copy of the generator hardware (i.e. they satisfy goal 3 above). This cannot be said of pseudo-random sequences generated by LFSRs even when operated in combinatorial arrangements. Various attacks can be successfully mounted against such methods.

Recommendation:

Simulcrypt control word generators should preferably use a physical source such as thermal noise, diode noise, MISC or the equivalent to generate random sequences. Pseudorandom techniques should be used advisedly and only after exhaustive testing to insure at least criteria 1 and 2 are met.

C.4 Control word randomness verification testing

There are numerous tests for randomness that can be applied to sequences, however in practical implementations there are two fundamental tests that can be relied upon to detect any significant defect in both pseudo- and true random sequences.

C.4.1 1/0 bias

The 1/0 bias test is usually performed on a sequence of convenient length and the comparator is trimmed to produce a logical 0 probability, $p(0)$, of 0,5.

$$p(0) = 0,5 + e \pm 0,001$$

where e = bias factor

XORing bits together will exponentially converge to:

$$p(0) = 0,5$$

A two-bit example:

$$p(0) = (0,5 + e)^2 + (0,5 - e)^2 = 0,5 + 2e^2$$

A four-bit example:

$$p(0) = 0,5 + 8e^4$$

Recommendation:

1/0 bias detection tests should be run on the generated sequences, and corrections should be made when needed.

C.4.2 Autocorrelation

Autocorrelation is defined as a discernible relationship in the variation of a variable over time. In order for a random sequence to be non-deterministic, its autocorrelation property has to be minimized. There are many algorithms available to measure autocorrelation properties, and most specify the test to be run on small (100 kbit/s) blocks where actual values should not vary more than three standard deviations from expected values for two consecutive blocks. Depending on the required speed, the tests may be run continuously or at frequent intervals.

Recommendation:

Autocorrelation tests should be run on sequences at intervals sufficient to ensure with reasonable certainty that no deterministic properties exist.

C.5 Testing locations

Although it is recommended that the above tests be conducted at the output of the control word generator, CA operators should consider conducting similar tests at the input of their ECMGs to confirm the randomness of the control words they receive.

Annex D (informative): Security Method for the SCS \leftrightarrow ECMG Interface

The following is a recommended method for encrypting the clear control word data traversing non-secure networks between the SCS and ECMG devices. For simplicity, it is performed at the application level within the Simulcrypt protocol. Since the CW data consists of an 8-byte block transferred over the interface once per CP, it is quite straightforward to encrypt using a standard block encryption algorithm. Key management as specified here is both simple and effective, requiring minimal resources. This method is facilitated by using the *CW_encryption* parameter and its sub-parameters as specified in the *CW_provision* message format.

D.1 Algorithm Selection

Although the *algorithm_type* parameter in the *CW_encryption* parameter allows the selection of multiple encryption algorithms, it is recommended that the head-end/uplink operator and the external CA provider(s) agree beforehand on the algorithm to be used. The *algorithm_type* parameter is also useful in specifying the key entropy (i.e. 40 bit vs. 56 bit) used with a particular algorithm where external CA systems are capable of both versions. Weakened key strengths are sometimes necessary to satisfy the requirements of governmental authorities. In the 40 bit case, two padding bytes of value 0x00 should replace bytes 5 and 6 of the selected key as shown below.

If the selected key value is: 9B F2 74 A0 B1 9A E6

Then the 40 bit adjusted key is: 00 00 74 A0 B1 9A E6

It is the responsibility of the Simulcrypting participants to select an algorithm that is strong enough, appropriate to this application, and compliant with national restrictions. Examples of algorithms that may be suitable can be found in FIPS 46-2 [19].

The encryption mode specified is Electronic Code Book and can be used with any 56 bit block cipher. Encryption and decryption are shown in figures D.1 and D.2 respectively.

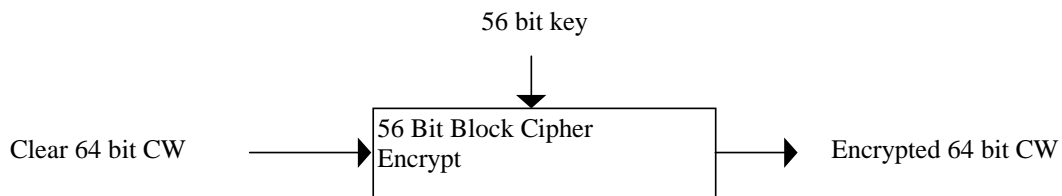


Figure D.1: Control Word encryption (SCS head-end/uplink function)

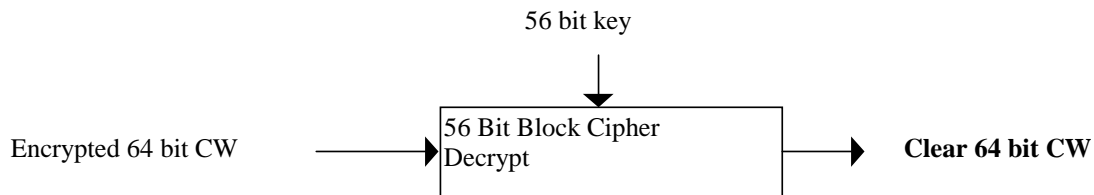


Figure D.2: Control Word decryption (External ECMG function)

D.2 Control Word processing

Only the 8-byte control word data field in the *CP_CW_combination* parameter is subject to encryption. The control word data is parsed from the 2-byte CP data field prior to encryption and is re-concatenated with it following encryption. This maintains an 8-byte plaintext field and requires only a single iteration of the encryption algorithm. Moreover, additional processing steps such as padding are avoided. Figure D.3 illustrates the parsing operation.

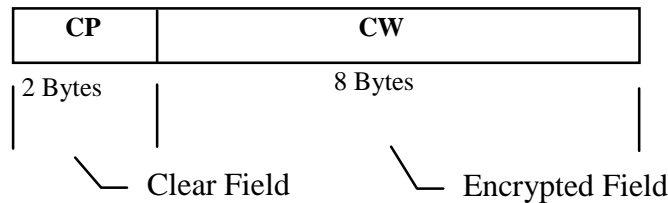


Figure D.3: *CP_CW_combination* field parsing

SCS devices need only implement the encryption mode of the algorithm, while ECMG devices need only implement the decryption mode.

D.3 Key Management

Under the present document, key management involves the generation, selection, and distribution of key data to be used in the algorithm for CW encryption. This has to be done in a standardized way in order to insure interoperability between SCS and ECMG devices.

D.3.1 Key Generation/Distribution

The encryption algorithm implementation in the present document uses key data generated by a good random source (see annex C) and stored on media for use in both SCS and ECMG devices. Each SCS (head-end/uplink) operator is responsible for securely generating the key data for his SCS \leftrightarrow ECMG interface(s) and for securely distributing it upon request to all external CA operators for use in their ECMGs. In the event a key list is suspected or known to be compromised, the SCS operator is responsible for generating and distributing a replacement list as soon as possible. Moreover, SCS operators may agree to generate and distribute new key lists on a periodic basis to insure key security. The suggested size of the key-space is 2 048 bytes. This provides 292 possible 7-byte keys per key list for use in the algorithm.

Highly-cautious SCS operators may wish to detect and exclude weak and semi-weak keys when generating key-lists, however due to the nature of the application, the occasional use of these keys does not pose a security threat. Moreover, if the random source is robust, neither weak nor semi-weak keys are likely to be generated with any significant probability.

To facilitate seamless transition from one key list to another, two independent 2 048-bit lists are used. Both the active list and the future list should reside on both the SCS and all Simulcrypting ECMGs before the head-end/uplink facility performs the transition function. The list in current use should be identified using the most significant bit in the *CW_encryption* parameter; this bit is designated as *key_list_sel* (key list select bit). When reset (0), the *A* key list is in use; when set (1), the *B* list is in use. Transition from one key list to the other is accomplished by setting or resetting the *key_list_sel* bit.

In order to avoid placing headers in the key lists, the SCS and ECMG software should examine the *key_list_sel* bit at the time lists are loaded to determine whether the list being loaded is to be designated the *A* or *B* list. If the active list is *A*, the list is loaded as *B*; if the active list is *B*, it is loaded as *A*. If no lists are active (during initialization) the software should allow the operator to manually enter the designator.

D.3.2 Selection

Selection of the 56-bit key data for use in the algorithm is accomplished by using a randomly-generated 11-bit key select vector. This vector is used as an index into the 2 048 key space as shown in figure D.4.

Only every seventh address is legal as a key select vector (i.e. 0, 7, 14, 21, etc.) through 2 037. This provides 292 possible completely independent keys.

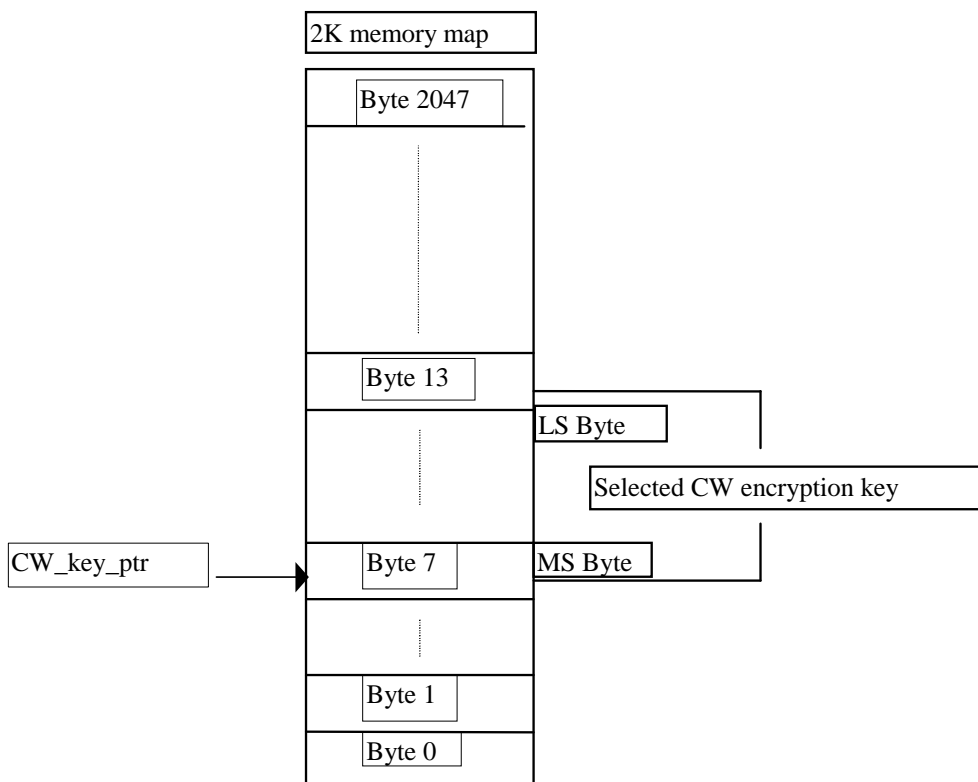


Figure D.4: Example of Control Word encryption key selection

The *CW_encryption* parameter is represented as a two-byte field as shown below. Bits 0 to 10 comprise the *CW_key_pointer*, and bits 11 to 13 are used to designate the cryptographic algorithm or key size in use. Bit 14 is used to invoke fixed key mode, and bit 15 is the A/B key list designator (*key_list_sel*).

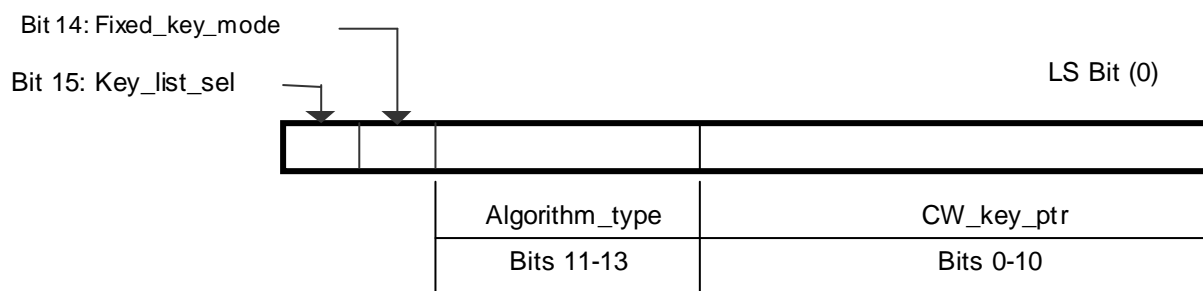


Figure D.5: CW_encryption Parameter (two bytes)

New keys are selected for each instance of the *CW_provision* message, and where more than one CW is conveyed in a single *CW_provision* message, all CWs are encrypted using the same key. One key is required per each CW encryption or decryption operation.

In the event 40 bit keys are in use, the padding method previously described is applied to the selected key data.

D.3.3 Key Pointer Distribution

The present document uses a symmetric algorithm, which is the same key is used for both encryption and decryption. Therefore both the SCS and ECMG should use the same key as selected by the *CW_key_ptr* parameter within the *CW_encryption* parameter. This parameter is generated in the SCS and should be conveyed in the *CW_provision* message immediately following the *CP_CW_combination* parameter. Clause 5.5.7 specifies the required *CW_encryption* parameter. The entire table is reproduced below for convenience; it includes the sub-parameters specific to this security method.

Parameter	Number of instances in message
<i>ECM_channel_id</i>	1
<i>ECM_stream_id</i>	1
<i>CP_number</i>	1
<i>CW_encryption</i> <i>CW_key_ptr</i> <i>Algorithm_type</i> <i>Fixed_key_mode</i> <i>Key_list_sel</i>	0 to 1
<i>CP_CW_combination</i>	<i>CW_per_msg</i>
<i>CP_duration</i>	0 to 1
<i>access_criteria</i>	0 to 1

CW_encryption: This parameter contains the four sub-parameters listed below that enable encrypting of control words over the SCS \leftrightarrow ECMG interface. If the parameter is included in the *CW_provision* message, control word scrambling is invoked; if omitted, CWs are being issued in the clear.

CW_key_ptr: This 11-bit field contains an index that points to the active CW encryption key contained on a 2 048 byte (or smaller) key list. It is a randomized value (1 of 292) generated within the SCS which points to the MS byte of a seven-byte key used in block cipher Electronic Code Book mode. Legal values include every 7th address in the 2 048 space (0, 7, 14, 21, etc).

algorithm_type: This field may be used either to signal the type of encryption algorithm in use for CW encryption or the key length of a given algorithm. It is useful where it may be desirable to change either the fundamental algorithm or its key length providing both the head-end and all external ECMGs have the appropriate capabilities. In most cases, the Simulcrypting participants will agree on these parameters in advance.

fixed_key_mode: This bit is used to bypass the key list and use a key contained in Read-Only Memory (ROM) for encryption of the control word. This key will need to be agreed upon by all Simulcrypting participants. The security method described in clause D.3.4 uses a defined fixed key value.

key_list_sel: In order to facilitate smooth changeover from one key list to another, two independent lists should be maintained on each ECMG and the SCS. This bit allows selection of one of the two lists as the active list.

D.3.4 Fixed Key Mode

There are instances in practical cryptosystems when it is desirable to temporarily encrypt messages under a common fixed key. This mode is useful when troubleshooting system failures, during initialization, or when the SCS has not installed a key list. It provides a fallback mode that is more secure than sending control words in the clear. When invoked, the SCS encrypts all CWs under the same fixed key, which is located in ROM and is not part of any key list. Fixed key mode is invoked using the *Fixed_key_mode* bit (14) of the *CW_encryption* parameter (see figure D.5). When set (1), the encryption key is selected from the appropriate (A or B) key list as designated by the pointer value and the key list select bit. When reset (0), the fixed key is used as the encryption key. The ECMG cannot invoke fixed key mode. The value of the fixed key is:

- 56-bit version: 4D A1 9F F0 AF 6B 8F;
- 40-bit version: 00 00 9F F0 AF 6B 8F.

It was generated in accordance with annex C. Since this mode can be invoked at any time by the SCS, the ECMG software should include an alarm to alert operators when fixed key mode is in effect.

D.4 Encryption Function Toggling

Although it is unlikely that either head-end operators or external CA providers will wish to discontinue CW encryption once it is invoked, there may be an occasional need to temporarily revert to clear CW transmission for system troubleshooting or for other reasons. This is accomplished by the SCS deleting the CW_encryption parameter from the CW_provision message. If the ECMG does not receive the CW_encryption parameter in the CW_provision message, it does not apply decryption to the received control words. Clear control should only be sent by prior arrangement between head-end operators and external CA providers or in emergencies; the ECMG does not have the capability to invoke clear CW transmission. ECMG designers may wish to include an alarm in their firmware that would activate upon detection of clear control word mode.

Annex E (normative):

Summary of Requirements for C(P)SIG \Leftrightarrow (P)SIG interface

This clause provides a high-level summary of the requirements imposed on the head-end system and each CAS in support of the C(P)SIG \Leftrightarrow (P)SIG interface.

The head-end and each CAS comply with the requirements presented in clauses E.1 and E.2, respectively, as well as all specifications referred to in clauses E.1 and E.2.

E.1 Head-end system requirements

The head-end shall be solely responsible for:

- 1) Generating and broadcasting one or more transport streams (TSs) that conform to MPEG-2 and DVB specifications (EN 300 468 [1] through ETR 289 [i.4] and ISO/IEC 13818-1 [3]).
- 2) Ensuring the MPEG-2 and DVB syntactic and semantic integrity of these TSs.
- 3) Ensuring that each TS include all standard MPEG-2 PSI and DVB SI tables that are required by MPEG-2 and DVB specifications (specifically, TR 101 154 [i.1] and TR 101 211 [i.3]). There is no requirement on the head-end to generate any standard tables that are optional (e.g. EIT Schedule).
- 4) Ensuring that each PSI and SI table (required or optional) include all descriptors required by MPEG-2 and DVB specifications, respectively (specifically, TR 101 154 [i.1] and TR 101 211 [i.3]), and optionally generating any other standard tables and/or descriptors defined by MPEG-2 and/or DVB. See also the next requirement.
- 5) Including, as required per DVB and CAS-specific conditional access (CA) requirements, CA_descriptors in all PSI CAT and PMT tables. However, the head-end shall not include any private_data_bytes in CA_descriptors whose CA_system_id belongs to any CAS that is interfaced with the head-end (refer to ISO/IEC 13818-1 [3]).
- 6) Optionally generating any private tables (i.e. with user-defined table_id) with MPEG-2 section syntax. (This is outside the scope of this interface. Commercial agreement should be used to avoid conflict with CAS-generated private tables transmitted on the PDG \Leftrightarrow MUX Interface).
- 7) Optionally generating, for its own purposes, an ordered list of private descriptors (i.e. with user-defined tag value) for insertion in any PSI/SI table; and broadcasting this list of descriptors. Either of two methods may be used to prevent conflict with private descriptors generated by any CAS (see clause E.2):
 - logically separating the head-end's private descriptors with a private_data_specifier descriptor, whose private_data_specifier value is used only by the head-end (per commercial agreement). The present document strongly recommends this approach; or
 - accordingly, the head-end may employ commercial agreement or some other unspecified means to prevent conflicts in private descriptor tag usage and interpretation.
- 8) Scheduling DVB SI services and events. While this is not specific to the C(P)SIG \Leftrightarrow (P)SIG Interface, the head-end shall inform the CAS(s) of new and/or changed services and events, per the "triggering" requirement presented below.
- 9) Configuration and initialization of the head-end (P)SIG processes defined in clause 8.2.1 of the present document. Specifically, either a single PSISIG, or a {PSIG+SIG} pair, shall be configured.
- 10) Hosting one or more CAS and their C(P)SIG processes, as defined in clause 8.2.1, and supporting each CAS per the requirements presented.
- 11) Maintaining mutual separation and independence of each CAS.

- 12) Triggering (signalling) each CAS before, or upon, the occurrence of any or all of the following actions:
 - new DVB SI EIT Following event;
 - new head-end information about a future DVB SI EIT event;
 - creation, modification or closure of an ECM stream;
 - user-defined (per commercial agreement);
 - in order to enable triggering, the CAS shall first tell the head-end which types of triggers it wants to receive, on a per-service basis. In addition, the CAS shall specify how far in advance of the action it wants to receive the trigger (if possible);
 - in this context, "service" means either a DVB-defined service or an MPEG-2 program, either being defined by a PAT program_number entry and a PMT section.
- 13) Fulfilling CAS requests for the insertion of a list of private descriptors in standard PSI and SI tables. The head-end shall therefore include any private descriptor requested by any CAS, provided that (a) the descriptor is syntactically valid, and (b) the integrity of the PSI/SI table(s) can be maintained. The descriptor list insertion may be synchronized with a triggered action (see above), or asynchronous, as requested by the C(P)SIG (see below).
- 14) Preventing conflict, among CASs and with the head-end, in the usage and interpretation of private descriptors. Either of two methods may be used:
 - logically separating each respective CAS's list with a private_data_specifier descriptor, whose private_data_specifier value is used only by the CAS (per commercial agreement). The present document strongly recommends this approach; or
 - employing commercial agreement, or some other unspecified means, to prevent conflicts in private descriptor tag usage and interpretation.
- 15) Fulfilling CAS requests to receive data from any PSI or SI currently transmitted. If the head-end generates EIT Schedule tables, they shall always be returned to the CAS in the clear, even if scrambled for broadcast (see EN 300 468 [1], clause 5.1.5).
- 16) Reporting specified error conditions per the above-defined interactions with the CASs.
- 17) Ensuring and maintaining all communications, networking, database access and so forth within the head-end, so as to ensure that all other requirements are met. Such intra-head-end interfaces are implementation-dependent, and outside the scope of the present document.

E.2 CAS's C(P)SIG requirements

Each CAS shall be solely responsible for:

- 1) Informing the head-end as to the types of event- and ECM-related triggers it wants to receive, on a per-service basis. In this context, "service" means either a DVB-defined service or an MPEG-2 program defined by a PAT program_number entry and a PMT section.
- 2) Processing action triggers received from the head-end.
- 3) Optionally generating, for its own purposes, an ordered list of private descriptors for insertion in any standard PSI/SI table(s) generated by the head-end:
 - each list supplied is associated with a CAS-specific private_data_specifier (per commercial agreement). It is the responsibility of the head-end to decide whether to separate descriptor lists via private_data_specifier descriptors, or by some other unspecified means. The present document strongly recommends the use of private_data_specifier descriptors;
 - the CAS may request that the descriptor list insertion be either synchronized with a triggered action, or asynchronous.

- 4) Sending the descriptor list to the head-end for broadcast:
 - a CAS cannot actually send or modify any PSI/SI table by itself. It can only request that the head-end insert its private descriptors in given tables.
- 5) Maintaining each descriptor list and each set of the CA_descriptor private_data_bytes up-to-date, per CAS requirements. The CAS shall ensure that private descriptors are compatible with the PSI/SI tables in which they are transmitted, at the time they are transmitted.
- 6) Reporting specified error conditions per the above-defined interactions with the head-end.
- 7) Ensuring and maintaining all communications, networking, database access and so forth within the CAS, so as to ensure that all other requirements are met. Such intra-CAS interfaces are implementation-dependent, and outside the scope of the present document.
- 8) Requesting for any PSI/SI table.

Annex F (informative): C(P)SIG \leftrightarrow (P)SIG Connection-oriented Configuration Example

The sample configuration presented here is referred to in clause 8.3.

Figure F.1 shows this reference configuration at the component and channel level. Figure F.2 depicts all connections at the stream level.

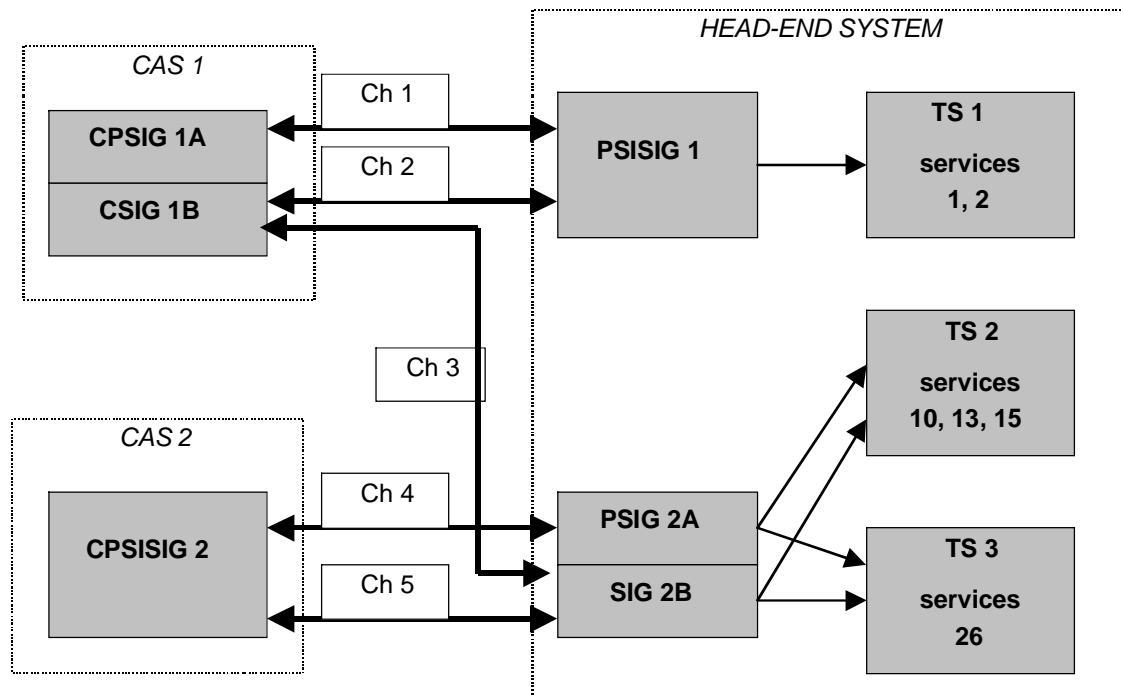


Figure F.1: Example: Channels (Ch 1 - Ch5) in a head-end system with two CASs

F.1 Head-end processes and configuration data

The head-end in this example includes two (P)SIGs. (P)SIG 1, a PSISIG, serves one transport stream, TS1. (P)SIG 2 consists of two processes, a PSIG and a SIG; they serve two transport streams, TS2 and TS3.

Tables F.1 and F.2 show the information that the head-end *might possibly* require to define the (P)SIG head-end processes (*recall that the present document does not define the format of this data*):

- table F.1 defines TS-related parameter;
- table F.2 defines (P)SIG-related parameters.

For simplicity, this data has been organized into relational tables in near-canonical form (some cells may contain lists of values). The following conventions are used in this and all other configuration tables presented in this annex:

Columns headed by names appearing in *ITALICS* indicate key fields. A specific system implementation might provide a different set of key fields for any table.

Cells marked with an asterisk (*) indicate data whose values are not of specific interest in understanding this example.

In addition, since the actual representation of this data is beyond the scope of the present document, the following are not to be inferred from this example:

- the completeness of this data: other data might be required;
- the location of this data (e.g. head-end and/or CAS); this would be determined by technical and/or commercial requirements.

Whether this data is static or dynamic; some of each would generally be required.

Table F.1: TS configuration data (example, not normative)

TRANSPORT STREAM	TRANSPORT_STREAM_id	ORIGINAL_NETWORK_id	NETWORK_ID	SERVICE_IDs
TS1	*	*	*	1, 2
TS2	*	*	*	10, 13, 15
TS3	*	*	*	26

Table F.2: (P)SIG configuration data (example, not normative)

(P)SIG NAME	IP ADDRESS	TCP PORT NO.	(P)SIG TYPE	TRANSPORT_STREAMs
PSISIG 1	*	*	3	TS1
PSIG 2A	*	*	1	TS2, TS3
SIG 2B	*	*	2	TS2, TS3

F.2 CAS processes and configuration data

Two CASs are hosted by the head-end. It is assumed that the head-end knows the CA_system_id (CASID) and private_data_specifier(s) used by each CAS.

The C(P)SIG of CAS 1 consists of two processes, a CPSIG and a CSIG. The C(P)SIG of CAS 2 is a CPSISIG.

Table F.3 shows the information that the head-end *might possibly* require to define the C(P)SIG CAS processes (*recall that the present document does not define the format of this data*).

Table F.3: C(P)SIG configuration data (example, not normative)

C(P)SIG NAME	CUSTOM_CAS_id		IP ADDRESS	TCP PORT NO.	C(P)SIG TYPE
	CASID	extension			
CPSIG 1A	CASID-1	1	*	*	4
CSIG 1B	CASID-1	2	*	*	8
CPSISIG 2	CASID-2	1	*	*	0xC

F.3 Channels and configuration data

Five channels are present in the system:

- channels *Ch1* and *Ch2* allow CAS 1 to request data from, and insert private data into, TS1. *Ch1* supplies MPEG-2 PSI private data for insertion, and *Ch2* similarly supplies DVB SI private data;

NOTE: CPSIG 1A may *request* DVB SI data via *Ch1*, and CSIG 1B may *request* MPEG-2 PSI data via *Ch2*. The kind of table data returned by the head-end via the table_response message is not restricted by the type of C(P)SIG that issued table_request. This principle holds, clearly, to all the other channels defined.

- similarly, channel *Ch3* allows CAS 1 to exchange data with TS2 and TS3. Only DVB SI private data is supplied for insertion;

- channels *Ch4* and *Ch5* allow CAS 2 to request data from, and insert private data into, TS2 and TS3. *Ch4* supplies MPEG-2 PSI private data, and *Ch5* similarly handles DVB SI private data.

Table F.4 shows the information that the head-end *might possibly* require to establish all the channel connections (*recall that the present document does not define the format of this data*).

Note the interpretation of the trigger_list values used:

- 0x0000003F** ("...00111111") says that the channel supports triggering on six kinds of actions: EIT future events, EIT following events, new ECM streams, ECM stream closure, ECM PID modification, and modification of ECM access criteria;
- 0x0000003D** ("...00111101") says that the channel supports triggering on all kinds of actions listed above, with exception of EIT future events.

Table F.4: Channel configuration data (example, not normative)

CHANNEL NAME	CUSTOM_CHANNEL_ID	C(P)SIG NAME	(P)SIG NAME	TRIGGER_LIST	MAX_STREAMS
Ch1	1	CPSIG 1A	PSISIG 1	0x3F	*
Ch2	2	CSIG 1B	PSISIG 1	0x3F	*
Ch3	3	CSIG 1B	SIG 2B	0x3F	*
Ch4	4	CPSISIG 2	PSIG 2A	0x3D	*
Ch5	5	CPSISIG 2	SIG 2B	0x3F	*

F.4 Streams and configuration data

Eight streams are defined in this sample configuration; they are depicted in figure F.2.

Table F.5 shows the information that the head-end *might possibly* require to establish all the stream connections (*recall that the present document does not define the format of this data*).

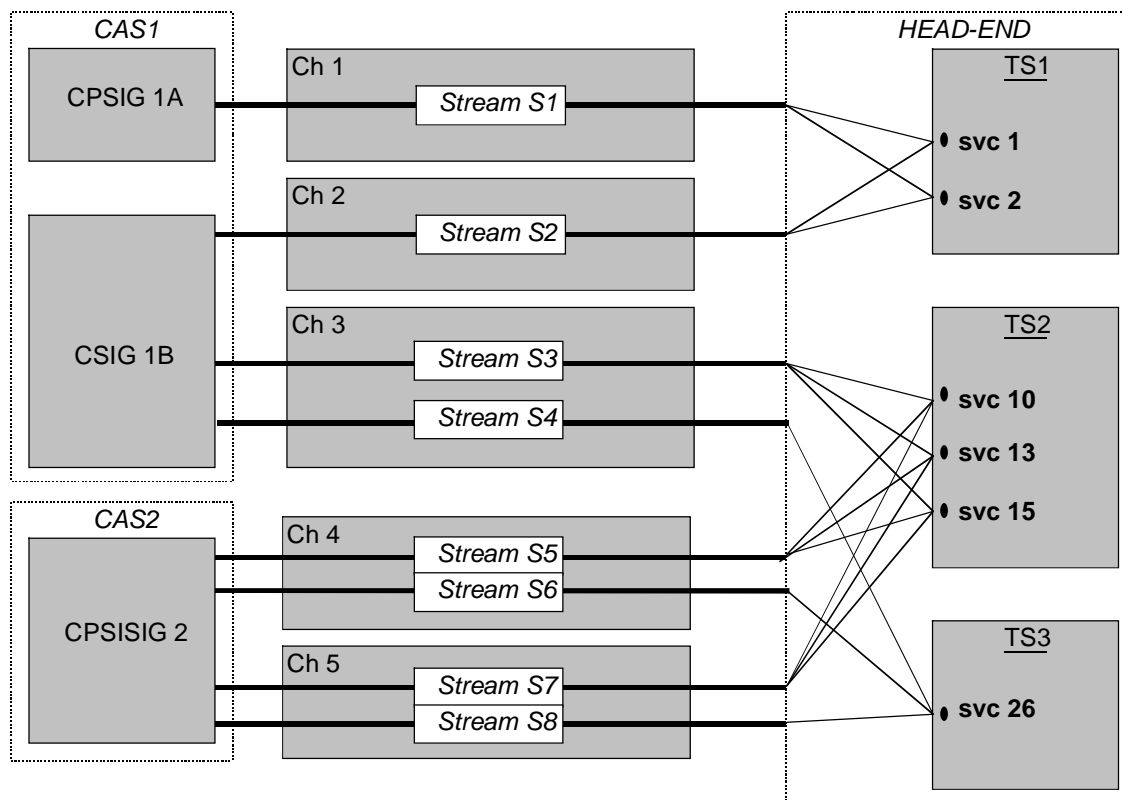


Figure F.2: Example: Stream interconnections for the example in figure F.1

Table F.5: Stream configuration data (*example, not normative*)

STREAM NAME	CUSTOM_ STREAM_id	CUSTOM_ CHANNEL_ID	TRANSPORT STREAM	SERVICE_ids	SERVICE_ PARAMETERS
S1	1	1	TS1	1, 2	*
S2	1	2	TS1	1, 2	*
S3	2	3	TS2	10, 13, 15	*
S4	3	3	TS3	26	*
S5	2	4	TS2	10, 13, 15	*
S6	3	4	TS3	26	*
S7	2	5	TS2	10, 13, 15	*
S8	3	5	TS3	26	*

Annex G (normative):

Transition Timing for EIS \Leftrightarrow SCS

The EIS may send an SCG provisioning message to the SCS using the optional parameter, *activation_time*.

The *activation_time* is passed in the UTC time-base and indicates when the Crypto-Period (CP) transition shall occur. The *activation_time* specifically denotes the change in scrambling. Each ECM stream transition shall occur at some reference to the CP transition depending on their timing delay offsets.

In the following examples, the tables depict state transition diagrams for a service that contains one content stream and two ECM streams; one ECM for CAS1 depicting a "current/next CW" model and one ECM for CAS2 depicting a "current CW" model.

ECM stream for CAS1:

Channel_status parameter	Value
<i>lead_CW</i>	1
<i>CW_per_msg</i>	2
<i>delay_start</i>	$x > 0$
<i>AC_delay_start</i>	$y < 0$
<i>Transition_delay_start</i>	$m < 0$
<i>Transition_delay_stop</i>	$n > 0$

ECM stream for CAS2:

Channel_status parameter	Value
<i>lead_CW</i>	0
<i>CW_per_msg</i>	1
<i>delay_start</i>	$x < 0$
<i>AC_delay_start</i>	$y < 0$
<i>transition_delay_start</i>	$m < 0$
<i>transition_delay_stop</i>	$n > 0$

A five second crypto-period duration is used for these examples. To simplify the examples, the *delay_stop* and *AC_delay_stop* values were not depicted.

Here the delay between the time when the *SCG_provision* message is received by the SCS and its requested activation time is assumed to be longer than the crypto-period.

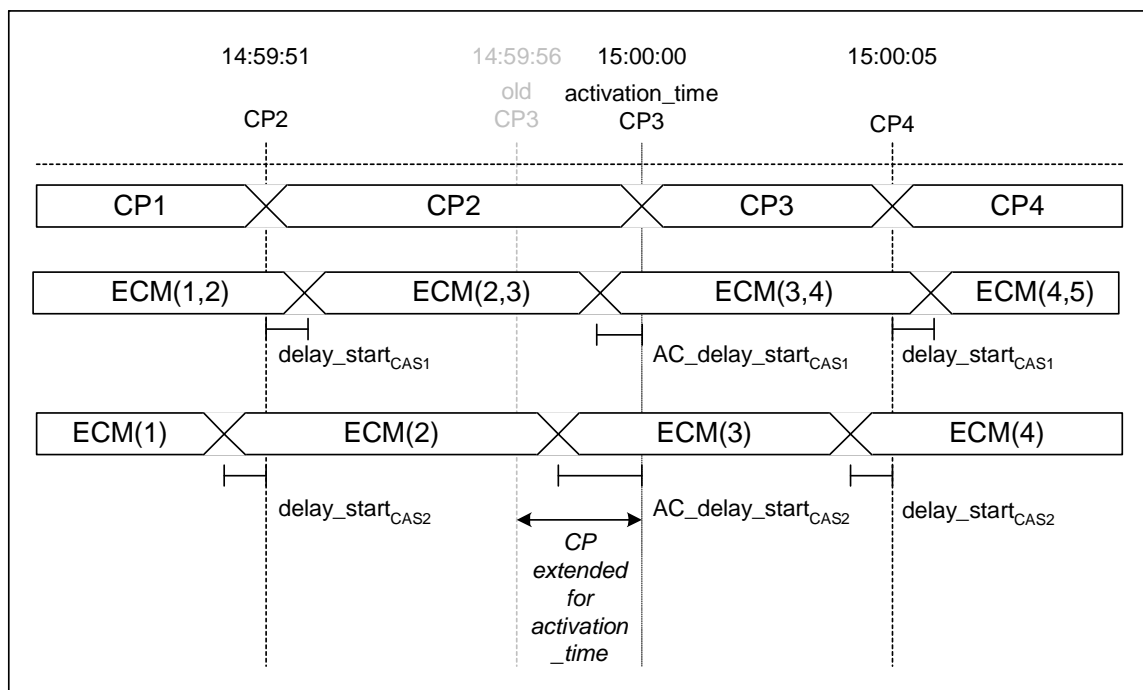


Figure G.1: Access Criteria Transition

Figure G.1 illustrates a transition in access criteria for both ECM streams at *activation_time* 15:00:00.000. In this example, crypto-period 2 (CP2) was extended to coincide with *activation_time*.

Since the access criteria for ECM CAS1 and CAS2 changed, the SCS shall use *AC_delay_start* in place of the default *delay_start* value. If the access criteria had not changed for one or both of the ECM streams, the SCS would have used the *delay_start* value even though CP2 was extended to the *activation_time*.

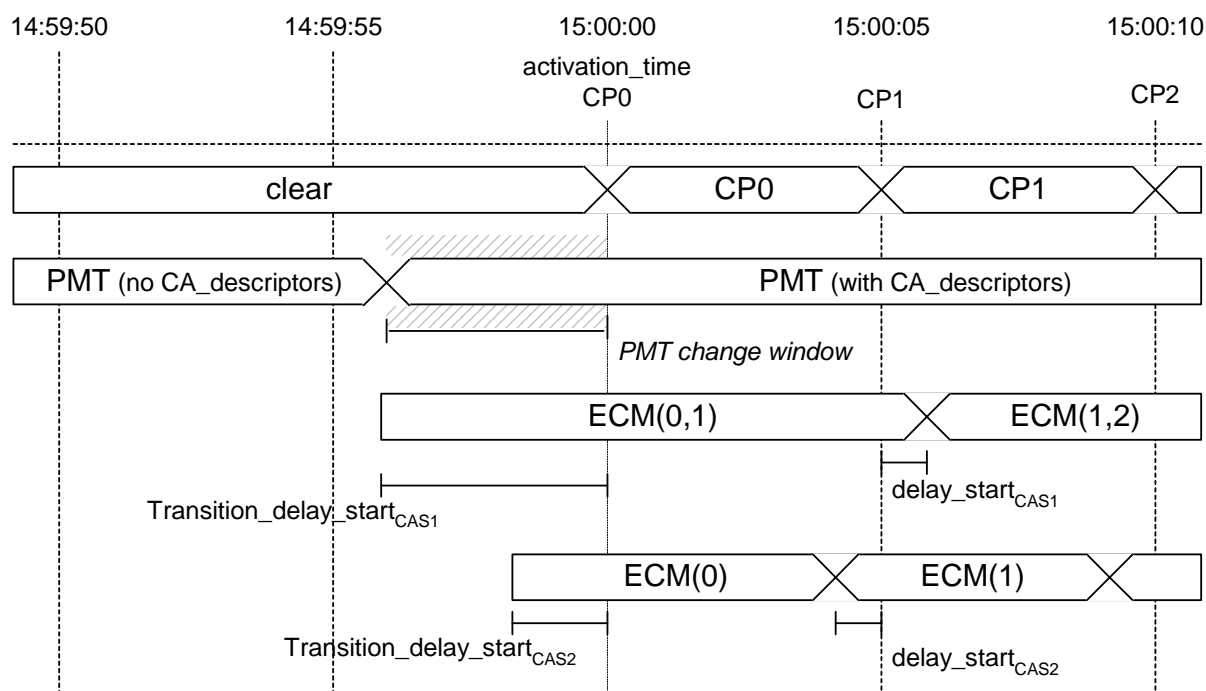


Figure G.2: Clear To Scramble Transition

Figure G.2 depicts a transition from clear to scramble at *activation_time* 15:00:00.000. This event occurs when the EIS sends an SCG provisioning message instructing the SCS to start scrambling one or more content streams and to start generating one or more ECM streams using the data defined in the ECM groups.

In this example, the ECM stream for CAS1 has a *transition_delay_start* that is less than 0. This allows the STB to receive the current CW (CW0) from the ECM before CP0. Otherwise, if *delay_start* were used, the STB could not descramble the content stream until CP1, which would be after the scrambler had already started scrambling.

The shaded area illustrates the ideal window for changing the PMT. The area is defined by the start of the ECM stream transmission for the ECM stream, within the SCG, that is starting last and the beginning of CP0.

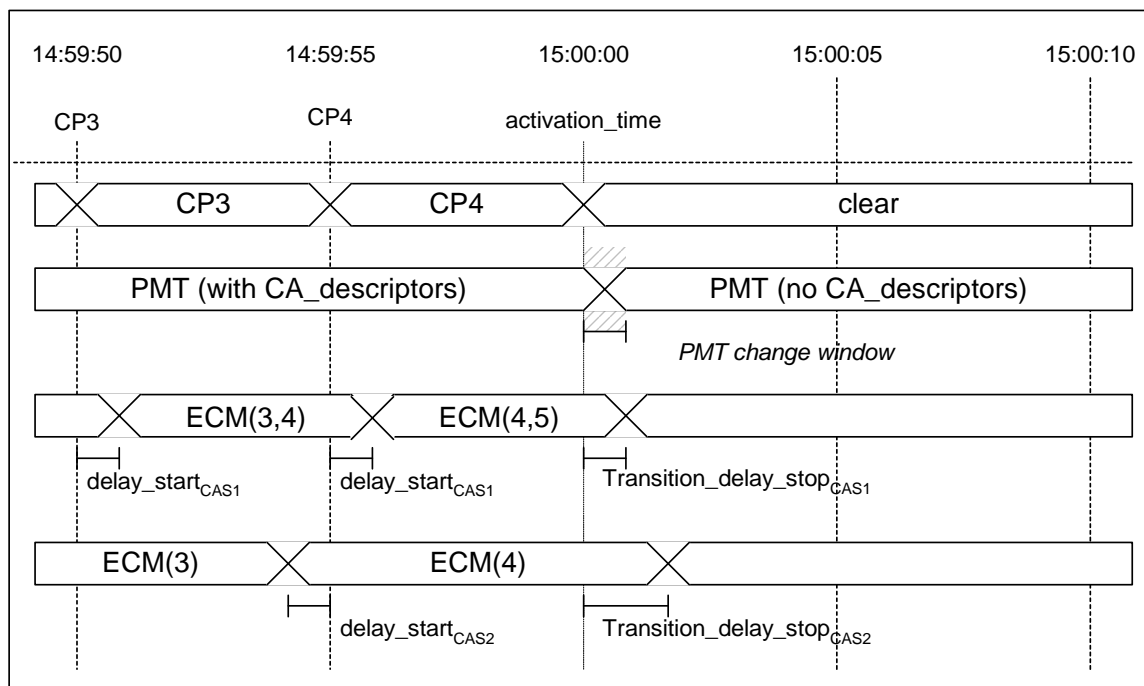


Figure G.3: Scramble To Clear Transition

Figure G.3 depicts a transition from scramble to clear at *activation_time* 15:00:00.000. This event occurs when the EIS sends an SCG provisioning message that no longer contains any content stream references (program numbers or elementary PIDs) or ECM groups.

The SCS shall transition to clear any content streams that are not provisioned in an SCG.

The SCS shall use the appropriate *transition_delay_stop* value for each ECM stream de-provisioned from an SCG group.

In this example, the ECM stream for CAS1 has a *transition_delay_stop* that is greater than 0. This allows the STB to continue to receive a valid ECM stream even though the service has gone clear. In the case of the ECM stream for CAS2, the ECM transition usually precedes the CP transition, but to ensure the STB can navigate the PMT (or equivalent) *transition_delay_stop* is greater than 0.

The shaded area illustrates the ideal window for changing the PMT. The area is defined by the start of the transition from scramble to clear and the end of the transmission of the first ECM stream, within the SCG.

Annex H (normative): Crypto-period duration management by the SCS

H.1 *Nominal_CP_duration* in ECMG ↔ SCS protocol

According to the standard, the nominal crypto-period duration for a particular SCG is determined by the SCS.

This value shall be chosen by the SCS:

- greater than all the *min_CP_duration* values specified at *channel_setup* by the ECMGs working with the SCS and involved by this SCG,
- greater than all the *max_comp_time* values of all the ECMGs involved by this SCG.

This means that any value of *nominal_CP_duration* complies with the standard if it meets both requirements above.

H.2 Management of the *recommended_CP_duration* value

The following clause applies only in the case when the SCS indicates in the *channel_status*, thanks to the *CP_duration_flag* that it is able to process the *recommended_CP_duration* value when provided in a *SCG_provision* message.

When the SCS receives a new *SCG_provision* message from the EIS which contains a *recommended_CP_duration* value, it shall determine the *nominal_CP_duration* for the SCG associated to this message as shown in figure H.1.

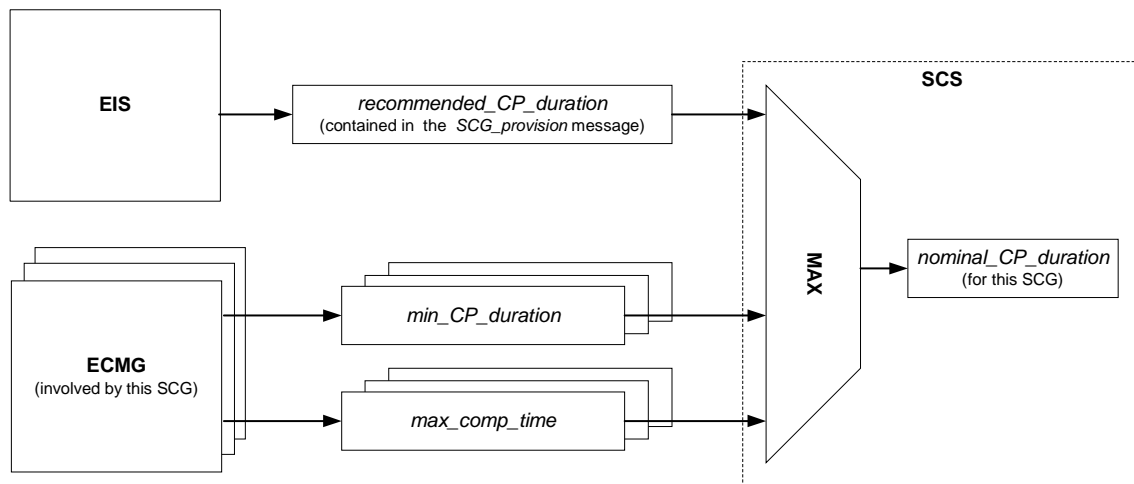


Figure H.1: Computation of the *nominal_CP_duration* value

Rules of computation of the *nominal_CP_duration* value associated to an SCG when receiving a new *SCG_provision* message related to this SCG, i.e.

$$\begin{aligned}
 \text{Nominal_CP_duration} = & \text{MAX}(\text{recommended_CP_duration}, \\
 & \text{MAX}(\text{min_CP_duration})_{\text{all involved ECMGs}}, \\
 & \text{MAX}(\text{max_comp_time})_{\text{all involved ECMGs}} \\
 &)
 \end{aligned}$$

Moreover, when the *recommended_CP_duration* value is provided for an existing SCG currently managed by the SCS, this latter shall apply one of the following policies to the ECM streams still associated to this SCG:

- The SCS defines the new *nominal_CP_duration* value as explained above. If the *nominal_CP_duration* value is modified the SCS closes all concerned streams on ECMG interface and opens again these streams with the new *nominal_CP_duration* parameter.
- the SCS defines the CP duration complying with the *recommended_CP_duration* according to figure H.1 and then in each following *CW_provision* messages provides the *CP_duration* parameter in using the obtained value.

To conclude, if no *recommended_CP_duration* is provided by the EIS in a new *SCG_provision* message, the SCS shall apply one of the following rules:

- if the *SCG_provision* message relates to a new SCG, the SCS shall use a default *recommended_CP_duration* value and compute the *nominal_CP_duration* associated to this SCG as depicted in figure H.1. The actual value of this default value is implementation dependant and is out of the scope of the present document.
- if the *SCG_provision* message relates to a SCG currently managed by the SCS, the SCS shall use the current *nominal_CP_duration* value as the *recommended_CP_duration* value and compute with it the new *nominal_CP_duration* as depicted in figure H.1.

Annex I (normative): Standard compliance

I.1 Overview

Three standards define the head-end implementation of DVB Simulcrypt:

- TS 101 197 (V1.2.1): "DVB Simulcrypt - Head-end architecture and synchronization" [21];
- TS 103 197 (V1.2.1): "Head-End Implementation of DVB Simulcrypt" [22];
- TS 103 197 (V1.3.1): "Head-End Implementation of DVB Simulcrypt" [25];
- the present document.

All these documents describe connection-based protocols.

According to the *protocol_version* parameter in each message:

- TS 101 197 [21] describes the ECMG protocol and the EMMG protocol in version 1 ("V1");
- TS 103 197 v1.2.1 [22] describes the ECMG protocol, the EMMG protocol and the C(P)SIG protocol in version 2 ("V2");
- TS 103 197 v1.3.1 [25] describes the ECMG protocol, the EMMG protocol, the C(P)SIG protocol, the (P)SIG \leftrightarrow MUX protocol and the EIS \leftrightarrow SCS protocol in version 3 ("V3");
- TS 103 197 (V1.4.1) [32] describes the (P)SIG \leftrightarrow MUX, the EIS \leftrightarrow SCS, the ACG, and the SIMCOMP \leftrightarrow MUXCONFIG protocol in version 4 ("V4"). All other interface remain on version 3 ("V3").
- The present document describes enhancements to the ECMG \leftrightarrow SCS and EMMG \leftrightarrow MUX protocols in version 5 ("V5") specifically for use with IP Datacasting over DVB-H Service Purchase and Protection [26]. The protocol version number for all other interfaces remains unchanged.

Compliance between V1 and V2 is described in DVB Simulcrypt (TR 102 035 [i.6]). This annex describes the compliance between V2 and V3, and between V3 and V4. Compliance between V1 and V3 is the combination of V1/V2 compliance and V2/V3 compliance.

The differences between V4 and V5 protocols for the ECMG \leftrightarrow SCS and EMMG \leftrightarrow MUX interfaces are described in annex N. These interfaces do not have compliance issues, as the V5 protocols are specifically for use in an IP Datacasting over DVB-H environment.

There is no compliance issue in a ACG \leftrightarrow EIS protocol implementation nor in an SIMCOMP \leftrightarrow MUXCONFIG protocol implementation because these protocols are described for the first time in TS 103 197 (V1.4.1) [32] and are not modified in the present document.

However, compliance issues can occur in a SCS/ECMG pair, in a MUX/EMMG pair, in a C(P)SIG/P(S)IG pair, in an EIS/SCS pair, and in a (P)SIG/MUX pair.

Besides connection-based protocols compliance between standard versions concerns the SIMF. Compliance issues can occur in CAS device MIB/NMS pair according to the standard version they refer to.

1.2 General compliance scheme for connection-based protocols

ECMG protocol, EMMG/PDG protocol (connection-based version), C(P)SIG protocol (connection-based version), the EIS \leftrightarrow SCS protocol, and the (P)SIG \leftrightarrow MUX protocol are concerned here and each is represented in a generic way as a "Client \leftrightarrow Server protocol".

In such a Client \leftrightarrow Server protocol, V_i and V_{i+1} are not compliant because of the management of the *protocol_version* parameter value. A V_i protocol accepts only the value i for the *protocol_version* parameter in header of messages and responses. Otherwise an "unsupported protocol version" error occurs.

The following general compliance scheme applies for all protocol versions where $i + 1 \leq 4$. No compliance issues exist for protocol V5, as this protocol is only for use in an IP Datacasting over DVB-H environments.

Basically:

- a $V_i + V_{i+1}$ or $V_{i+1} + V_i$ configuration in a Client/Server pair shall be avoided because it cannot work.
- a $V_i + V_i$ configuration or a $V_{i+1} + V_{i+1}$ configuration in a Client/Server pair is recommended.

If the Client is compliant with V_i and V_{i+1} , the Client connects the Server in V_{i+1} mode. If the Server is V_i -compliant, the following policy is recommended:

- Such a V_i -Server shall generate an V_i error message ("Unsupported protocol version").
- Then the Client disconnects, and connects again the Server in V_i mode.

If the Server is compliant with V_i and V_{i+1} , regardless of whether the version of the Client connecting to it is i or $i+1$, the following policy is recommended:

- Such a V_i and V_{i+1} compliant Server selects its current version according to the version detected in the *channel_setup* message received from the Client.
- This selection shall be performed for each channel, to allow a V_i -Client and a V_{i+1} -Client to connect the same Server.

Protocol	Client	Server
ECMG \leftrightarrow SCS	SCS	ECMG
EMMG/PDG \leftrightarrow MUX	EMMG/PDG	MUX
C(P)SIG \leftrightarrow (P)SIG	(P)SIG	C(P)SIG
EIS \leftrightarrow SCS	EIS	SCS
(P)SIG \leftrightarrow MUX	(P)SIG	MUX

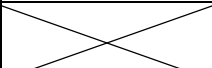
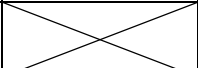
	V_i -Client	V_{i+1} -Client	V_i/V_{i+1} -Client	
V_i -Server	OK		Client $\rightarrow V_i$	Protocol version switch by Client if error on Channel_Setup
V_{i+1} -Server		OK	Client $\rightarrow V_{i+1}$	
V_i/V_{i+1} -Server	Server $\rightarrow V_i$	Server $\rightarrow V_{i+1}$	V_i or V_{i+1}	
	Protocol version is detected by Server at Channel_Setup		(same as \leftarrow)	

Figure I.1: V_i/V_{i+1} compliance

I.3 Functional difference between V2 and V3 in ECMG ↔ SCS protocol

There is only one functional difference between V2 and V3 of the ECMG protocol:

Crypto-period shortening in case of event alignments and error handling: a V2-SCS shall not shorten the current crypto-period length, a V3-SCS may shorten the current crypto-period length within the limits of *min_CP_duration* and *max_comp_time* values.

I.4 Functional differences between V2 and V3 in EMMG ↔ PDG protocol

There is no functional difference between V2 and V3 of the EMMG/PDG protocol.

I.5 Functional differences between V2 and V3 in C(P)SIG ↔ (P)SIG protocol

In V3 the C(P)SIG can be triggered by the (P)SIG on Private Data events. In particular see clauses 8.2.3, 8.2.5, 8.2.7, 8.3.2 and 8.3.4.11.

I.6 SIMF

Even if the protocol version is not significant in MIB, in both following clauses "V2" and "V3" are used to refer to MIB described in TS 103 197 [22] and in the present document respectively.

I.6.1 Functional differences between V2 and V3

V3 MIB includes the following differences:

- The individual SEM and SLM modules are optional. See clauses 7.2, 7.2.4.4 and 7.2.5.3.
- SIM ECMG Channel Table, SIM EMMG Channel Table and SIM EMMG Lap Table include new entries. See tables 18, 21 and 23, respectively.
- The new PD Trigger Table supports Private Data Event Trigger in transaction-based version of the C(P)SIG protocol. See clause 8.4.3.6.

I.6.2 Recommendation for SIMF compliance

In the Simulcrypt Identification Module (see table 16), the *simMibVersion* entry written by the CAS should allow the NMS for identifying the version of MIB supported by the CAS device.

A V2 + V3 or V3 + V2 configuration in a CAS device/NMS pair shall be avoided because it cannot work, except in the following case: if a V3-MIB compliant CAS device includes both SEM and SLM modules it can be addressed by a V2-MIB compliant NMS or by a V3-MIB compliant NMS.

A V2 + V2 configuration or a V3 + V3 configuration in a CAS device/NMS pair is recommended.

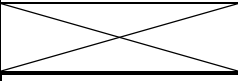
	V2-CAS device	V3-CAS device
V2-NMS	OK	OK only if CAS device supports SEM and SLM
V3-NMS		OK
V2/V3-NMS	NMS → V2	NMS → V3
If MIB version can be detected by NMS in <i>simMibVersion</i> entry		

Figure I.2: V2/V3 SIMF compliance

I.7 Functional differences between V3 and V4 in EIS⇔SCS protocol

All references to the *network_id* parameter in V3 should be replaced with *original_network_id* parameter in V4.

I.8 Functional differences between V3 and V4 in (P)SIG⇔MUX protocol

All references to the *network_id* parameter in V3 should be replaced with *original_network_id* parameter in V4.

I.9 Functional differences between V4 and V5 in ECMG⇔SCS and EMMG⇔MUX protocols

The V5 protocols for ECMG⇔SCS and EMMG⇔MUX are intended to support the Service Purchase and Protection mechanism associated with IP Datacasting over DVB-H [26].

The enhancements to the ECMG⇔SCS interface are meant to provide better support for the three AES-based scrambling algorithms that replace the DVB-CSA algorithm in an IP Datacasting over DVB-H configuration.

The optional enhancement to the EMMG⇔MUX interface allows EMMs/KMMs to be delivered to an IP Encapsulator using the same DVB protocol that is used for delivering EMMs to a multiplexer.

Refer to annex N for further information on the V5 enhancements to the ECMG⇔SCS and EMMG⇔MUX protocols.

Annex J (informative):

Use of DVB ASI for the PSIG \Leftrightarrow MUX interface

DVB has developed an interface specification for conveying data as transport stream packets from one piece of equipment to another. The specification, standardized by CENELEC as EN 50083-9 [20], has three methods, the Synchronous Parallel Interface (SPI), Synchronous Serial Interface (SSI), and Asynchronous Serial Interface (ASI). In practice, the ASI version is the most flexible and convenient of the three types of interface, and the ASI interface has consequently been very widely implemented on DVB multiplexing equipment both for data input and output. The use of ASI for the data flow is an option in the DVB Simulcrypt specifications for the (P)SIG \Leftrightarrow MUX interface. The interface has the following characteristics:

- 1) the interface is widely available, including on legacy multiplexing equipment;
- 2) the interface can handle a wide range of data rates;
- 3) the packet timing is readily preserved, to the precision required for SI and PSI data.

The PSI and/or SI contribution from an individual (P)SIG to a specific multiplexer should consist of all the required data for a specific set of PIDs, so that the MUX is only required to perform a multiplexing operation at the packet level.

In a system configuration, it may be convenient for the (P)SIG to supply the (P)SI packet data via ASI using different packet_IDs from those used in the output multiplex, and make use of the multiplexer's ability to change the packet_ID value. For example, different versions of the data for a particular destination SI packet_ID are needed for different multiplexes (for packet_IDs 0x0011 and 0x0012), but could be supplied from the same SIG over the same ASI signal.

The present document mandates the use of a TCP/IP connection between the (P)SIG and the MUX for control purposes. The control interface manages connection set-up, closure, packet_ID allocation, and bandwidth negotiation. The same messages are applicable, whether the data content is supplied via TCP/IP or via ASI.

It is possible to operate an ASI connection from a (P)SIG to a MUX without a TCP/IP control link in place. In this case, there is no standard way of controlling the connection or data-flow. However, in some distributed system architectures, with a (P)SIG remote from a MUX, there may only exist a forward path from the (P)SIG to the MUX. Provided that the system needs for PSI or SI are not dynamically changing in terms of packet_ID allocation or bandwidth, a static arrangement with independent configuration of the (P)SIG and the MUX may suffice. However, considerable care is needed in the configuration to ensure that the packet_ID allocations and bandwidth usage are consistent. Errors in configuration may be in particular difficult to locate.

Annex K (normative): ASN.1 MIBs description

K.1 SIM MIB

```

SIM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Unsigned32, Counter32, IPAddress,
    BITS FROM SNMPv2-SMI
    TEXTUAL-CONVENTION, RowStatus, DisplayString FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF;

simMIB MODULE-IDENTITY
    LAST-UPDATED "9707021700Z"
    ORGANIZATION "DVB Simulcrypt Technical Group"
    CONTACT-INFO " --- "
    DESCRIPTION
        "The MIB module for defining DVB Simulcrypt Conditional
        Access System configuration information."
    ::= { 1 3 6 1 4 1 2696 1 1}

AdministrativeState ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Administrative state as defined by ITU-T Recommendation X.734"
    SYNTAX BITS
    {
        locked(0),
        unlocked(1),
        shuttingDown(2)
    }

CaDescInsMode ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Conditional Access Descriptor Insertion Type."
    SYNTAX BITS
    {
        PsigInsertion(0),
        NoPsigInsertion(1)
    }

DelayType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Delay type."
    SYNTAX BITS
    {
        immediate(0),
        synchronized(1)
    }

```

```

DescriptorStatus ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The return status of descriptor insertion."
    SYNTAX BITS
    {
        success(0),
        unknownTrigger(1),
        unknownLocation(2),
        unsupportedDelay(3),
        unknownContext(4),
        unknownOtherTS(5),
        unknownNetwork(6),
        unknownTS(7),
        unknownES(8),
        unknownBouquet(9),
        unknownEvent(10),
        tableNotSupported(11),
        tableFull(12),
        other(13)
    }

ECMGChannelId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The ECM_channel_ID is represented as a 2 bytes unsigned integer."
    SYNTAX INTEGER (0..65535)

ECMGDelayValue ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The value of xxx_Delay_Start / _Stop in ECMG protocol."
    SYNTAX Signed16

ECMTriggerType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The type of an ECM trigger in a PSI generator."
    SYNTAX BITS
    {
        ecmStreamOpen(0),
        ecmStreamClose(1),
        ecmStreamChange(2),
        accessCriteriaChange(3)
    }

EMMGChannelId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the Data Channel Id."
    SYNTAX INTEGER (0..65535)

EMMGCommCapability ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Type of communications capability between EMMG/PDG and Multiplexer:
        TCP or UDP or both."
    SYNTAX BITS
    {
        both(0),
        tcp(1),
        udp(2)
    }

EMMGCommType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Type of communications capability between EMMG/PDG and Multiplexer:
        TCP or UDP."
    SYNTAX BITS
    {
        tcp(0),
        udp(1)
    }

```

```

EMMGDataType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Type of data carried in the EMMG/PDG Multiplexer stream."
    SYNTAX BITS
    {
        emm(0),
        other(1)
    }

FlowId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The flow identifier is represented as a 2 bytes unsigned integer."
    SYNTAX INTEGER (0..65535)

FlowType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Type of Flow: EMM, ECM, or private data."
    SYNTAX BITS
    {
        ecm(0),
        emm(1),
        privatedata(2)
    }

InsertLocation ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Descriptor insertion location."
    SYNTAX BITS
    {
        pmtLoop1(0),
        pmtLoop2(1),
        cat(2),
        nitLoop1ActualNet(3),
        nitLoop2ActualNet(4),
        nitLoop1OtherNet(5),
        nitLoop2OtherNet(6),
        batLoop1(7),
        batLoop2(8),
        sdtActualTS(9),
        sdtOtherTS(10),
        eitPFAcualTS(11),
        eitPFOtherTS(12),
        eitScheduleActualTS(13),
        eitScheduleOtherTS(14)
    }

ProvTableId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Provision table identifier."
    SYNTAX BITS
    {
        pat(0),
        cat(1),
        pmt(2),
        nitActualNet(3),
        nitOtherNet(4),
        bat(5),
        sdtActualTS(6),
        sdtOtherTS(7),
        eitPFAcualTS(8),
        eitPFOtherTS(9),
        eitScheduleActualTS(10),
        eitScheduleOtherTS(11)
    }

```

```

PsigType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Psig type."
    SYNTAX BITS
    {
        sig(0),
        psig(1),
        psisig(2)
    }

SectionTSPktFlag ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Format of ECM, EMM, Private data datagrams: section or TS packet."
    SYNTAX BITS
    {
        section(0),
        tspacket(1),
    }

StreamId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The Stream_ID is represented as a 2 bytes unsigned integer."
    SYNTAX INTEGER (0..65535)

SuperCasId ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "SuperCASId / Client_ID: a unsigned 32-bit identifier."
    SYNTAX Unsigned32

TriggerType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The type of a trigger in a PSI generator."
    SYNTAX BITS
    {
        dvbEvent(0),
        futureDvbEvent(1),
        newEcmStream(2),
        flowPidChange(3),
        accessCriteriaChange(4),
        ecmStreamClosure(5),
        pdStreamCEvent(6)
    }

simMIBObjects      OBJECT IDENTIFIER ::= { simMIB 1 }
simMIBConformance OBJECT IDENTIFIER ::= { simMIB 2 }

simIdent          OBJECT IDENTIFIER ::= { simMIBObjects 1 }
simECMG           OBJECT IDENTIFIER ::= { simMIBObjects 2 }
simEMMG          OBJECT IDENTIFIER ::= { simMIBObjects 3 }
simCPSI           OBJECT IDENTIFIER ::= { simMIBObjects 4 }
simPSI            OBJECT IDENTIFIER ::= { simMIBObjects 5 }

-- -----
-- Ident Group - This group is used for software configuration management
-- of all Simulcrypt components and includes the following objects:
--

simSoftwareVersion OBJECT-TYPE
    SYNTAX DisplayString (SIZE (80))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This contains a display string that defines the current version
        of the software for this unit."
    ::= { simIdent 1 }

```

```

simMIBVersion OBJECT-TYPE
    SYNTAX DisplayString (SIZE (80))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This contains a display string that defines the current version
        of the MIB."
    ::= { simIdent 2}

simMIBPrivateVersion OBJECT-TYPE
    SYNTAX DisplayString (SIZE (80))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This contains a display string that defines the current private
        version of the MIB."
    ::= { simIdent 3}

simAgentVersion OBJECT-TYPE
    SYNTAX DisplayString (SIZE (80))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This contains a display string that defines the current version
        of the agent."
    ::= { simIdent 4 }

-- -----
-- ECM Generator Group - This group is used for configuration management and status
-- monitoring of ECM Generators. It identifies each one of the ECM Generators by the
-- IP Address and TCP/UDP Port Number. It also associates Super_CAS_IDs, ECM_Channel_IDs,
-- and ECM_Stream_IDs with ECM Generators. It also associates status information and
-- statistics with channels and streams. The ECM Generator Group consists of three
-- conceptual tables. The first table is the interconnection table and is used for
-- the Headend Network Manager to query the IP addresses and the port number to be used
-- by an SCS to create a channel. It is indexed by a unique EcmgIndex which is an integer
-- assigned by the -- ECMG agent:
--

simEcmgTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEcmgEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table specifies the IP addresses and Port numbers of ECM Generators
        to be used by headend managers to configure SCSs. This table is to be
        used in ECM Generators and ECM Generator proxies."
    ::= { simECMG 1 }

simEcmgEntry OBJECT-TYPE
    SYNTAX SimEcmgEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. Depending on whether this is
        an ECMG agent or ECMG proxy agent different table entries can be omitted."
    INDEX { simEcmgIndex}
    ::= { simEcmgTable 1 }

SimEcmgEntry ::= SEQUENCE {
    simEcmgIndex          INTEGER (0..65535),
    simEcmgIpAddress      IPAddress,
    simEcmgTcpPort        INTEGER (0..65535),
    simEcmgSuCasId        SuperCasId,
    simEcmgChannels       Counter32,
    simEcmgCwPrs          Counter32,
    simEcmgErrs           Counter32,
    simEcmgTargetCpsig    INTEGER (0..65535),
    simEcmgCaMib          OBJECT IDENTIFIER
}

```

```

simEcmgIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The ECM Generator Table unique index."
    ::= { simEcmgEntry 1 }

simEcmgIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "IP address of the host of the ECMG."
    ::= { simEcmgEntry 2 }

simEcmgTcpPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "TCP port of the ECMG."
    ::= { simEcmgEntry 3 }

simEcmgSuCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The Super_VAS_ID is formed by concatenation of the CA_system_id
        (16 bit) and the CA_subsystem_ID (16 bit). It defines uniquely a
        set of ECMGs for a given SCS."
    ::= { simEcmgEntry 4 }

simEcmgChannels OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of channels this ECMG is currently maintaining."
    ::= { simEcmgEntry 5 }

simEcmgCwPrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of CW provisioning requests received by this ECMG."
    ::= { simEcmgEntry 6 }

simEcmgErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors for this ECMG."
    ::= { simEcmgEntry 7 }

simEcmgTargetCpsig OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The index into the C(P)SIG table identifying the C(P)SIG associated
        with this ECMG."
    ::= { simEcmgEntry 8 }

simEcmgCaMib OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The pointer to a provider proprietary MIB (like ifSpecific in
        the interfaces group of MIB II."
    ::= { simEcmgEntry 9 }

```

```

-- -----
-- ECMG Channel Table - Used for monitoring channel information. It is indexed
-- by the ECMG Index from the ECMG table and the ChannelId.
--

simEcmgCTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEcmgCEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table specifies information relating to ECMG/SCS channels including
        the IP addresses and Port numbers of SCSs communicating
        with the ECMG Generators. "
    ::= { simECMG 2 }

simEcmgCEntry OBJECT-TYPE
    SYNTAX SimEcmgCEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. Depending on whether this is
        an ECMG agent or ECMG proxy agent different table entries can be omitted."
    INDEX { simEcmgIndex, simEcmgChannelId}
    ::= { simEcmgCTable 1 }

SimEcmgCEntry ::= SEQUENCE {
    simEcmgChannelId      ECMGChannelId,
    simEcmgCSsIpAddress   IpAddress,
    simEcmgCSsTcpPort     INTEGER (0..65535),
    simEcmgCStreams       Counter32,
    simEcmgCCwPrs         Counter32,
    simEcmgCErrs          Counter32,
    simEcmgCSuCasId       ECMGCSuCasId,
    simEcmgFormat         SectionTSPktFlag,
    simACDelayStart       ECMGDelayValue,
    simACDelayStop        ECMGDelayValue,
    simDelayStart         ECMGDelayValue,
    simDelayStop          ECMGDelayValue,
    simTransitionDelayStart ECMGDelayValue,
    simTransitionDelayStop ECMGDelayValue,
    simECMRepPeriod       INTEGER (0..65535),
    simMaxStreams         Counter 32,
    simMinCPDuration      INTEGER (0..65535),
    simLeadCW             Counter 32,
    simCWPerMsg           Counter 32,
    simMaxCompTime        INTEGER (0..65535)
}

simEcmgChannelId OBJECT-TYPE
    SYNTAX ECMGChannelId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The ECMG/SCS Channel identifier."
    ::= { simEcmgCEntry 1 }

simEcmgCSsIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "IP address of the SCS."
    ::= { simEcmgCEntry 2 }

simEcmgCSsTcpPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "TCP port of the SCS."
    ::= { simEcmgCEntry 3 }

```



```

simEcmgCStreams OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of streams this ECMG is currently maintaining on this channel."
    ::= { simEcmgCEntry 4 }

simEcmgCCwPrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of CW provisioning requests received by this ECMG on this channel."
    ::= { simEcmgCEntry 5 }

simEcmgCErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors for this ECMG on this channel."
    ::= { simEcmgCEntry 6 }

simEcmgCSuCasId OBJECT-TYPE
    SYNTAX ECMGCSuCasId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The SuperCASId of the current Channel."
    ::= { simEcmgCEntry 7 }

simEcmgFormat OBJECT-TYPE
    SYNTAX SectionTSPktFlag
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Format of datagrams : section or TS Packet."
    ::= { simEcmgCEntry 8 }

simACDelayStart OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of AC_Delay_Start parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 9 }

simACDelayStop OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of AC_Delay_Stop parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 10 }

simDelayStart OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of Delay_Start parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 11 }

simDelayStop OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of Delay_Stop parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 12 }

```

```

simTransitionDelayStart OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of Transition_Delay_Start parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 13 }

simTransitionDelayStop OBJECT-TYPE
    SYNTAX ECMGDelayValue
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Value of Transition_Delay_Stop parameter in ECMG protocol, imposed by CAS."
    ::= { simEcmgCEntry 14 }

simECMRepPeriod OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Repeating period of ECM defined by CAS and applied by SCS."
    ::= { simEcmgCEntry 15 }

simMaxStreams OBJECT-TYPE
    SYNTAX Counter 32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Max number of streams supported by the ECMG."
    ::= { simEcmgCEntry 16 }

simMinCPDuration OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Min crypto-period length supported by the ECMG."
    ::= { simEcmgCEntry 17 }

simLeadCW OBJECT-TYPE
    SYNTAX Counter 32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of CW the ecmg needs in advance."
    ::= { simEcmgCEntry 18 }

simCWPerMsg OBJECT-TYPE
    SYNTAX Counter 32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of CW the ecmg needs in each ECM request."
    ::= { simEcmgCEntry 19 }

simMaxCompTime OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Max delay supported by the ECMG for providing an ECM."
    ::= { simEcmgCEntry 20 }

-- -----
-- ECMG Stream Table - Used for monitoring stream information. It is indexed
-- by the ECMG Index from the ECMG table, the ChannelId from the Channel Table
-- and the StreamId.
--

simEcmgSTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEcmgSEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table specifies information relating to ECMG/SCS streams. "
    ::= { simECMG 3 }

```

```

simEcmgSEntry OBJECT-TYPE
    SYNTAX SimEcmgSEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. Depending on whether this is
        an ECMG agent or ECMG proxy agent different table entries can be omitted."
    INDEX { simEcmgIndex, simEcmgChannelId, simEcmgStreamId }
    ::= { simEcmgSTable 1 }

SimEcmgSEntry ::= SEQUENCE {
    simEcmgStreamId      StreamId,
    simEcmgEcmId         FlowId,
    simEcmgSLastCp       Unsigned32,
    simEcmgSCwPrs        Counter32,
    simEcmgSErrs         Counter32
}

simEcmgStreamId OBJECT-TYPE
    SYNTAX StreamId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The ECMG/SCS Stream identifier."
    ::= { simEcmgSEntry 1 }

simEcmgEcmId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The unique ECM flow identifier."
    ::= { simEcmgSEntry 2 }

simEcmgSLastCp OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The number of the crypto-period last processed on this stream."
    ::= { simEcmgSEntry 3 }

simEcmgSCwPrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of CW provisioning requests received by this ECMG on
        this stream."
    ::= { simEcmgSEntry 4 }

simEcmgSErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors for this ECMG on this stream."
    ::= { simEcmgSEntry 5 }

-- -----
-- EMM/PD Generator Group - This group is used for management of EMM/PD Generators.
-- It identifies each one of the EMM/PD Generators by the IP Address and TCP/UDP Port
-- Number. It also associates Client_IDs, Data_stream_IDs, and Data_Channel_IDs with EMM/PD
-- Generators. It also associates status information and statistics with streams. The
-- EMMG/PDG Generator Group consists of four conceptual tables. The first table is used
-- for information relevant to EMMG/PDG and is indexed by a unique EmOrPdIndex which is
-- assigned by the EMMG/PDG agent:
--
simEmOrPdTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEmOrPdEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table defines the EMMG or PDG interfaces to the MUX and is to be
        used in EMMGs/PDGs and optionally the multiplexer."
    ::= { simEMMG 1 }

```

```

simEmOrPdEntry OBJECT-TYPE
    SYNTAX SimEmOrPdEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. Depending on whether this is an
        EMMG/PDG or multiplexer agent different table entries can be omitted."
    INDEX { simEmOrPdIndex }
    ::= { simEmOrPdTable 1 }

SimEmOrPdEntry ::= SEQUENCE {
    simEmOrPdIndex      INTEGER (0..65535),
    simEmOrPdDataType   EMMGDataType,
    simEmOrPdClientId   SuperCasId,
    simEmOrPdCommCapability EMMGCommCapability,
    simEmOrPdErrs       Counter32,
    simEmOrPdTargetCpsig INTEGER (0..65535),
    simEmOrPdCaMib       OBJECT IDENTIFIER
}

simEmOrPdIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Unique index into the EMMG or PDG table. "
    ::= { simEmOrPdEntry 1 }

simEmOrPdDataType OBJECT-TYPE
    SYNTAX EMMGDataType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Data_type: Type of data handled by this EMMG/PDG. "
    ::= { simEmOrPdEntry 2 }

simEmOrPdClientId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Client_ID: The Client_ID is a 32-bit identifier. It shall identify uniquely
        an EMMG/PDG across all the EMMGs/PDGs connected to a given MUX. To facilitate
        uniqueness of this value, the following rules apply:
        * In the case of EMMs or other CA related data, the two first bytes of the
        client_id should be equal to the two bytes of the corresponding CA_system_ID.
        * In other cases a value allocated by DVB for this purpose should be used."
    ::= { simEmOrPdEntry 3 }

simEmOrPdCommCapability OBJECT-TYPE
    SYNTAX EMMGCommCapability
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Communication capability between EMMG/PDG and the multiplexer. Currently
        TCP or UDP or both."
    ::= { simEmOrPdEntry 4 }

simEmOrPdErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors for this EMMG/PDG."
    ::= { simEmOrPdEntry 5 }

simEmOrPdTargetCpsig OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The index into the C(P)SIG table identifying the C(P)SIG associated
        with this EMMG/PDG."
    ::= { simEmOrPdEntry 6 }

```

```

simEmOrPdCaMib OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Pointer to a vendor proprietary extension to the EMMG/PDG MIB group."
    ::= { simEmOrPdEntry 7 }

-- -----
-- EMMG/PDG Logical Access Point Table - The second EMM Generator/ PD Generator table
-- is used for configuration of the EMMGs/PDGs. It is uniquely indexed by the
-- EmOrPdLapIndex which is a globally assigned quantity (with respect to the headend)
-- and associates globally assigned Logical Access Points (LAPs) with mux ports.
--

simEmOrPdLapTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEmOrPdLapEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for configuration of EMM/PD Generators."
    ::= { simEMMG 2 }

simEmOrPdLapEntry OBJECT-TYPE
    SYNTAX SimEmOrPdLapEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simEmOrPdLapIndex }
    ::= { simEmOrPdLapTable 1 }

SimEmOrPdLapEntry ::= SEQUENCE {
    simEmOrPdLapIndex          INTEGER (0..65535),
    simEmOrPdLapAdminState     AdministrativeState,
    simEmOrPdLapCommType       EMMGCommType,
    simEmOrPdLapMuxIpAddress   IpAddress,
    simEmOrPdLapMuxPort        INTEGER (0..65535),
    simEmOrPdLapStatus         RowStatus
    simEmOrPdLapMuxUIpAddress   IpAddress,
    simEmOrPdLapMuxUPort       INTEGER (0..65535),
}

simEmOrPdLapIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Unique Logical Access Point (LAP) identifier. "
    ::= { simEmOrPdLapEntry 1 }

simEmOrPdLapAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Used by an authorized manager to lock a conceptual row for exclusive
        write and create access."
    ::= { simEmOrPdLapEntry 2 }

simEmOrPdLapCommType OBJECT-TYPE
    SYNTAX EMMGCommType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Type of communication between EMMG/PDG and the multiplexer. Currently TCP or UDP."
    ::= { simEmOrPdLapEntry 3 }

simEmOrPdLapMuxIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "IP address of the multiplexer for EMMG/PDG TCP communication."
    ::= { simEmOrPdLapEntry 4 }

```

```

simEmOrPdLapMuxPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Port number (TCP) of the multiplexer for EMMG/PDG TCP communication."
    ::= { simEmOrPdLapEntry 5 }

simEmOrPdLapStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used for table row creation management."
    ::= { simEmOrPdLapEntry 6 }

simEmOrPdLapMuxUIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "IP address of the multiplexer for EMMG/PDG UDP communication."
    ::= { simEmOrPdLapEntry 7 }

simEmOrPdLapMuxUPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Port number (UDP) of the multiplexer for EMMG/PDG UDP communication."
    ::= { simEmOrPdLapEntry 8 }

-- -----
-- EMMG/PDG Logical Access Point Group Table - The third EMM Generator/ PD Genartor table is used
-- for
-- configuration of the EMMGs/PDG. It associates LAP Groups and LAPs and is
-- uniquely indexed by the EmOrPdLapGroup, and EmOrPdLapIndex.
--

simEmOrPdLapGTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEmOrPdLapGEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for configuration of EMM/PD Generators."
    ::= { simEMMG 3 }

simEmOrPdLapGEntry OBJECT-TYPE
    SYNTAX SimEmOrPdLapGEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simEmOrPdLapGroup, simEmOrPdLapIndex }
    ::= { simEmOrPdLapGTable 1 }

SimEmOrPdLapGEntry ::= SEQUENCE {
    simEmOrPdLapGroup      INTEGER (0..65535),
    simEmOrPdLapGAdminState AdministrativeState,
    simEmOrPdLapGStatus    RowStatus
}

simEmOrPdLapGroup OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Logical Access Point (LAP) group. "
    ::= { simEmOrPdLapGEntry 1 }

simEmOrPdLapGAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Used by an authorized manager to lock a conceptual row for exclusive
        write and create access."
    ::= { simEmOrPdLapGEntry 2 }

```

```

simEmOrPdLapGStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used for table row creation management."
    ::= { simEmOrPdLapGEntry 3 }

-- -----
-- EMMG/PDG Channel Table - Used for monitoring of EMM / PD Generator channels.
--

simEmOrPdCTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEmOrPdCEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for monitoring of channels between Muxes and EMMGs/PDGs."
    ::= { simEMMG 4 }

simEmOrPdCEntry OBJECT-TYPE
    SYNTAX SimEmOrPdCEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simEmOrPdIndex, simEmOrPdLapIndex, simEmOrPdChannelId }
    ::= { simEmOrPdCTable 1 }

SimEmOrPdCEntry ::= SEQUENCE {
    simEmOrPdChannelId      EMMGChannelId,
    simEmOrPdCommType       EMMGCommType,
    simEmOrPdCIpAddress     IpAddress,
    simEmOrPdCPort          INTEGER (0..65535),
    simEmOrPdCErrors        Counter32
    simEmOrPdCFormat        SectionTSPktFlag,
    simEmOrPdCUIpAddress    IpAddress,
    simEmOrPdCUIPort        INTEGER (0..65535),
    }

simEmOrPdChannelId OBJECT-TYPE
    SYNTAX EMMGChannelId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Data_channel_ID: This identifier uniquely identifies a
        EMM/Private Data channel within a client_ID."
    ::= { simEmOrPdCEntry 1 }

simEmOrPdCommType OBJECT-TYPE
    SYNTAX EMMGCommType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Communications type: TCP or UDP."
    ::= { simEmOrPdCEntry 2 }

simEmOrPdCIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "IP address of the host of the EMMG or PDG."
    ::= { simEmOrPdCEntry 3 }

simEmOrPdCPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Port number *TCP or UDP of the EMMG or PDG."
    ::= { simEmOrPdCEntry 4 }

```

```

simEmOrPdCErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors on this channel."
    ::= { simEmOrPdCEntry 5 }

simEmOrPdCFormat OBJECT-TYPE
    SYNTAX SectionTSPktFlag
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Format of datagrams : section or TS Packet."
    ::= { simEmOrPdCEntry 6 }

simEmOrPdCUIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "IP address of the host of the EMMG or PDG for UDP."
    ::= { simEmOrPdCEntry 7 }

simEmOrPdCUPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Port number (UDP) of the EMMG or PDG."
    ::= { simEmOrPdCEntry 8 }

-- -----
-- EMMG/PDG Stream Table - Used for monitoring of EMM / PD Generator streams.
--

simEmOrPdSTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimEmOrPdSEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for monitoring of streams between Muxes and EMMGs/PDGs."
    ::= { simEMMG 5 }

simEmOrPdSEntry OBJECT-TYPE
    SYNTAX SimEmOrPdSEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simEmOrPdIndex, simEmOrPdLapIndex, simEmOrPdDataId }
    ::= { simEmOrPdSTable 1 }

SimEmOrPdSEntry ::= SEQUENCE {
    simEmOrPdDataId      FlowId,
    simEmOrPdSChannelId  EMMGChannelId,
    simEmOrPdBwidth      Unsigned32,
    simEmOrPdStreamId    StreamId,
    simEmOrPdSErrs       Counter32,
    simEmOrPdSBytes      Counter32,
    simEmOrPdSReqBwidth  Unsigned32,
}

simEmOrPdDataId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "DataID: This identifier uniquely identifies a EMM/Private Data stream."
    ::= { simEmOrPdSEntry 1 }

```



```

simEmOrPdSChannelId OBJECT-TYPE
    SYNTAX EMMGChannelId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Channel identifier."
    ::= { simEmOrPdSEntry 2 }

simEmOrPdBwidth OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Negotiated Bandwidth."
    ::= { simEmOrPdSEntry 3 }

simEmOrPdStreamId OBJECT-TYPE
    SYNTAX StreamId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Data_stream_ID: This identifier uniquely identifies a EMM/Private
        Data stream within a channel."
    ::= { simEmOrPdSEntry 4 }

simEmOrPdSErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communications errors on this stream."
    ::= { simEmOrPdSEntry 5 }

simEmOrPdSBytes OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of bytes sent by this EMMG/PDG on this stream."
    ::= { simEmOrPdSEntry 6 }

simEmOrPdReqBwidth OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Bandwidth as requested by CAS."
    ::= { simEmOrPdSEntry 7 }

-- -----
-- C(P)SIG) Group - This Group is used for management of some aspects of inteaction
-- between the custom PSI Generators (C(P)SIG)) and the PSI Generator. It consists of
-- three tables. The first table is used for advertising C(P)SIG) information by the
-- C(P)SIG) host. The second table is used for the manager to configure the C(P)SIG).
-- The third and fourth tables are used for C(P)SIG) channel and stream monitoring.
--

simCpsigTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimCpsigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table defines the C(P)SIG) interfaces to the Mux and is to be
        used in the C(P)SIG and optionally the multiplexer."
    ::= { simCPSI 1 }

simCpsigEntry OBJECT-TYPE
    SYNTAX SimCpsigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. Depending on whether this is an
        EMMG/PDG or multiplexer agent different table entries can be omitted."
    INDEX { simCpsigIndex }
    ::= { simCpsigTable 1 }

```

```

SimCpsigEntry ::= SEQUENCE {
    simCpsigIndex      INTEGER (0..65535),
    simCpsigSuperCasId SuperCasId,
    simCpsigErrs       Counter32,
    simCpsigChannels   Counter32,
    simCpsigCpsigIpAddress IpAddress,
    simCpsigCpsigPort   INTEGER(0..65535),
    simCpsigCaMib       OBJECT IDENTIFIER
}

simCpsigIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Unique index into the Cpsig table. "
    ::= { simCpsigEntry 1 }

simCpsigSuperCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "super_CAS_id "
    ::= { simCpsigEntry 2 }

simCpsigErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communication errors for this C(P)SIG)."
    ::= { simCpsigEntry 3 }

simCpsigChannels OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of channels for this C(P)SIG)."
    ::= { simCpsigEntry 4 }

simCpsigCpsigIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The IP Address of the C(P)SIG)."
    ::= { simCpsigEntry 5 }

simCpsigCpsigPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The TCP port number of the C(P)SIG)."
    ::= { simCpsigEntry 6 }

simCpsigCaMib OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        " Pointer to a vendor proprietary extension to the C(P)SIG) MIB group."
    ::= { simCpsigEntry 7 }

-- -----
-- C(P)SIG) Channel Table - Used for monitoring of C(P)SIG channels.
--

simCpsigCTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimCpsigCEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for monitoring of channels between (P)SIGs and C(P)SIGs."
    ::= { simCPSI 2 }

```

```

simCpsigEntry OBJECT-TYPE
    SYNTAX SimCpsigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        " Information about a single table entry."
    INDEX { simCpsigIndex, simCpsigChannelId }
    ::= { simCpsigCTable 1 }

SimCpsigEntry ::= SEQUENCE {
    simCpsigChannelId      INTEGER (0..65535),
    simCpsigPsigIpAddress  IpAddress,
    simCpsigPsigPort       INTEGER (0..65535),
    simCpsigCErrs          Counter32,
    simCpsigCTstrms        Counter32,
    simCpsigCSstrms        Counter32
}

simCpsigChannelId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "ChannelId identifies the C(P)SIG channel."
    ::= { simCpsigEntry 1 }

simCpsigPsigIpAddress OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "IP address of the host of the (P)SIG."
    ::= { simCpsigEntry 2 }

simCpsigPsigPort OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "TCP Port number of the (P)SIG."
    ::= { simCpsigEntry 3 }

simCpsigCErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communication errors on this channel."
    ::= { simCpsigEntry 4 }

simCpsigCTstrms OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of transport streams on this channel."
    ::= { simCpsigEntry 5 }

simCpsigCSstrms OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of service streams on this channel."
    ::= { simCpsigEntry 6 }

```

```

-----
-- C(P)SIG Stream Table - Used for monitoring of C(P)SIG streams.
--

simCpsigStreamTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimCpsigStreamEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is used for monitoring of streams between Muxes and C(P)SIGs."
    ::= { simCPSI 3 }

simCpsigStreamEntry OBJECT-TYPE
    SYNTAX SimCpsigStreamEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simCpsigIndex, simCpsigChannelId, simCpsigStreamId }
    ::= { simCpsigStreamTable 1 }

SimCpsigStreamEntry ::= SEQUENCE {
    simCpsigStreamId      INTEGER (0..65535),
    simCpsigStreamTStreamId  INTEGER (0..65535),
    simCpsigStreamNid     INTEGER (0..65535),
    simCpsigStreamOnid    INTEGER (0..65535),
    simCpsigStreamMaxCompTime  INTEGER (0..65535),
    simCpsigStreamTriggerEnable  TriggerType,
    simCpsigStreamLastTrigger   TriggerType,
    simCpsigStreamLastEventId   INTEGER (0..65535),
    simCpsigStreamLastServiceId  INTEGER (0..65535),
    simCpsigStreamLastEsId     INTEGER (0..65535),
    simCpsigStreamLastEcmPid    INTEGER (0..65535),
    simCpsigStreamErrs         Counter32,
    simCpsigStreamBytes        Counter32
}

simCpsigStreamId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This identifier uniquely identifies a C(P)SIG stream"
    ::= { simCpsigStreamEntry 1 }

simCpsigStreamTStreamId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This identifier uniquely identifies a C(P)SIG transport stream"
    ::= { simCpsigStreamEntry 2 }

simCpsigStreamNid OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This identifier uniquely identifies the network ide associated with the stream"
    ::= { simCpsigStreamEntry 3 }

simCpsigStreamOnid OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This identifier uniquely identifies the original network ide associated with
        the stream"
    ::= { simCpsigStreamEntry 4 }

simCpsigStreamMaxCompTime OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Max Computation time by the C(P)SIG."
    ::= { simCpsigStreamEntry 5 }

```

```

simCpsigStreamTriggerEnable OBJECT-TYPE
    SYNTAX TriggerType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Triggers enabled by the C(P)SIG."
    ::= { simCpsigStreamEntry 6 }

simCpsigStreamLastTrigger OBJECT-TYPE
    SYNTAX TriggerType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Last trigger processed by the C(P)SIG."
    ::= { simCpsigStreamEntry 7 }

simCpsigStreamLastEventId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Last event id processed by the C(P)SIG."
    ::= { simCpsigStreamEntry 8 }

simCpsigStreamLastServiceId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Last service id processed by the C(P)SIG."
    ::= { simCpsigStreamEntry 9 }

simCpsigStreamLastEsId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Last elementary stream id processed by the C(P)SIG."
    ::= { simCpsigStreamEntry 10 }

simCpsigStreamLastEcmPid OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Last ECM pid processed by the C(P)SIG."
    ::= { simCpsigStreamEntry 11 }

simCpsigStreamErrs OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of communication errors on this stream."
    ::= { simCpsigStreamEntry 12 }

simCpsigStreamBytes OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The total number of bytes sent by this EMMG/PDG on this stream."
    ::= { simCpsigStreamEntry 13 }

-- -----
-- (P)SIG Group - This Group is used for the synchronization and information
-- exchange between the PSI Generator and Custom PSI Generators and
-- between the SI Generator and Custom SI Generators.
--

simPsigTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table advertises the (P)SIG configuration information. "
    ::= { simPSI 1 }

```

```

simPsigEntry OBJECT-TYPE
    SYNTAX SimPsigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigIndex }
    ::= { simPsigTable 1 }

SimPsigEntry ::= SEQUENCE {
    simPsigIndex          INTEGER (0..65535),
    simPsigType           PsigType,
    simPsigTriggerSupport TriggerType,
    simPsigNetworkId      INTEGER (0..65535),
    simPsigONetworkId     INTEGER (0..65535),
    simPsigTransStreamId  INTEGER (0..65535),
    simPsigTSServices     OCTET STRING (SIZE(0..511))
}

simPsigIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The unique index into the table."
    ::= { simPsigEntry 1 }

simPsigType OBJECT-TYPE
    SYNTAX PsigType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Psig type"
    ::= { simPsigEntry 2 }

simPsigTriggerSupport OBJECT-TYPE
    SYNTAX TriggerType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Identifies which trigger types the PSIG supports."
    ::= { simPsigEntry 3 }

simPsigNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Network identifier. "
    ::= { simPsigEntry 4 }

simPsigONetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Original Network identifier. "
    ::= { simPsigEntry 5 }

simPsigTransStreamId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Transport Stream identifier. "
    ::= { simPsigEntry 6 }

simPsigTSServices OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..511))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "List of service identifies on the transport stream. "
    ::= { simPsigEntry 7 }

--
--

```

```

simPsigConfigTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigConfigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table configures the (P)SIG/(C)PSIG communication. "
    ::= { simPSI 2 }

simPsigConfigEntry OBJECT-TYPE
    SYNTAX SimPsigConfigEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigConfigCustCasId, simPsigConfigIndex , simPsigIndex}
    ::= { simPsigConfigTable 1 }

SimPsigConfigEntry ::= SEQUENCE {
    simPsigConfigIndex      INTEGER (0..65535),
    simPsigConfigAdminState AdministrativeState,
    simPsigConfigCpsigType  PsigType,
    simPsigConfigCustCasId  SuperCasId,
    simPsigConfigMaxCompTime INTEGER(0..65535),
    simPsigConfigServiceId  INTEGER (0..65535),
    simPsigConfigTriggerEnable TriggerType,
    simPsigConfigCADInsMode CaDescInsMode,
    simPsigConfigEntryStatus RowStatus
}

simPsigConfigIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The unique index into the table."
    ::= { simPsigConfigEntry 1 }

simPsigConfigAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used by an authorized manager to lock a conceptual row for exclusive
        write and create access."
    ::= { simPsigConfigEntry 2 }

simPsigConfigCpsigType OBJECT-TYPE
    SYNTAX PsigType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "C(P)SIG type."
    ::= { simPsigConfigEntry 3 }

simPsigConfigCustCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Custom CAS Identifier."
    ::= { simPsigConfigEntry 4 }

simPsigConfigMaxCompTime OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Maximum Computing Time. "
    ::= { simPsigConfigEntry 5 }

simPsigConfigServiceId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Service identifier. "
    ::= { simPsigConfigEntry 6 }

```

```

simPsigConfigTriggerEnable OBJECT-TYPE
    SYNTAX TriggerType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Trigger types enabled. "
    ::= { simPsigConfigEntry 7 }

simPsigConfigCADInsMode OBJECT-TYPE
    SYNTAX CaDescInsMode
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Conditional Access Descriptor Insert mode. "
    ::= { simPsigConfigEntry 8 }

simPsigConfigEntryStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used for table row creation management."
    ::= { simPsigConfigEntry 9 }

--
--

simPsigEcmTrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigEcmTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the active ECM Triggers."
    ::= { simPSI 3 }

simPsigEcmTrEntry OBJECT-TYPE
    SYNTAX SimPsigEcmTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigEcmTrIndex }
    ::= { simPsigEcmTrTable 1 }

SimPsigEcmTrEntry ::= SEQUENCE {
    simPsigEcmTrIndex      INTEGER (0..65535),
    simPsigEcmTrNetworkId  INTEGER (0..65535),
    simPsigEcmTrONetworkId INTEGER (0..65535),
    simPsigEcmTrTransStreamId  INTEGER (0..65535),
    simPsigEcmTrServiceId    INTEGER (0..65535),
    simPsigEcmTrEsId        INTEGER (0..65535),
    simPsigEcmTrType        ECMTriggerType,
    simPsigEcmTrSuCasId     SuperCasId,
    simPsigEcmTrEcmId       FlowId,
    simPsigEcmTrEcmPid      INTEGER (0..65535),
    simPsigEcmTrAccessCriteria  OCTET STRING (SIZE(0..127))
}

simPsigEcmTrIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The trigger index."
    ::= { simPsigEcmTrEntry 1 }

simPsigEcmTrNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier."
    ::= { simPsigEcmTrEntry 2 }

```



```

simPsigEcmTrOnNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier."
    ::= { simPsigEcmTrEntry 3 }

simPsigEcmTrTransStreamId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier. "
    ::= { simPsigEcmTrEntry 4 }

simPsigEcmTrServiceId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Service Identifier. "
    ::= { simPsigEcmTrEntry 5 }

simPsigEcmTrEsId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Elementary Stream Identifier. "
    ::= { simPsigEcmTrEntry 6 }

simPsigEcmTrType OBJECT-TYPE
    SYNTAX ECMTriggerType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Trigger Type. "
    ::= { simPsigEcmTrEntry 7 }

simPsigEcmTrSuCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Client Identifier. "
    ::= { simPsigEcmTrEntry 8 }

simPsigEcmTrEcmId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Stream Identifier. "
    ::= { simPsigEcmTrEntry 9 }

simPsigEcmTrEcmPid OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM PID "
    ::= { simPsigEcmTrEntry 10 }

simPsigEcmTrAccessCriteria OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..127))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Access criteria. "
    ::= { simPsigEcmTrEntry 11 }

--
--

```

```

simPsigFlowTrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigFlowTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the active Flow Triggers."
    ::= { simPSI 4 }

simPsigFlowTrEntry OBJECT-TYPE
    SYNTAX SimPsigFlowTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigFlowTrIndex }
    ::= { simPsigFlowTrTable 1 }

SimPsigFlowTrEntry ::= SEQUENCE {
    simPsigFlowTrIndex      INTEGER (0..65535),
    simPsigFlowTrType       FlowType,
    simPsigFlowTrSuCasId    SuperCasId,
    simPsigFlowTrFlowId     FlowId,
    simPsigFlowTrFlowPID    INTEGER (0..65535)
}

simPsigFlowTrIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The trigger index."
    ::= { simPsigFlowTrEntry 1 }

simPsigFlowTrType OBJECT-TYPE
    SYNTAX FlowType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Flow Type. "
    ::= { simPsigFlowTrEntry 2 }

simPsigFlowTrSuCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Flow Super CAS identifier. "
    ::= { simPsigFlowTrEntry 3 }

simPsigFlowTrFlowId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Flow Stream Identifier. "
    ::= { simPsigFlowTrEntry 4 }

simPsigFlowTrFlowPID OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Flow PID "
    ::= { simPsigFlowTrEntry 5 }

--
--

simPsigEvtTrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigEvtTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the active EVNT Triggers."
    ::= { simPSI 5 }

```

```

simPsigEvtTrEntry OBJECT-TYPE
    SYNTAX SimPsigEvtTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigEvtTrIndex }
    ::= { simPsigEvtTrTable 1 }

SimPsigEvtTrEntry ::= SEQUENCE {
    simPsigEvtTrIndex      INTEGER (0..65535),
    simPsigEvtTrNetworkId  INTEGER (0..65535),
    simPsigEvtTrONetworkId INTEGER (0..65535),
    simPsigEvtTrTransStreamId INTEGER (0..65535),
    simPsigEvtTrServiceId  INTEGER (0..65535),
    simPsigEvtTrEventId    INTEGER (0..65535),
    simPsigEvtTrStartTime  OCTET STRING (SIZE (0..4)),
    simPsigEvtTrDuration   OCTET STRING (SIZE (0..2)),
    simPsigEvtTrPrivateData OCTET STRING (SIZE (0..256))
}

simPsigEvtTrIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The event trigger index."
    ::= { simPsigEvtTrEntry 1 }

simPsigEvtTrNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier."
    ::= { simPsigEvtTrEntry 2 }

simPsigEvtTrONetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier."
    ::= { simPsigEvtTrEntry 3 }

simPsigEvtTrTransStreamId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier. "
    ::= { simPsigEvtTrEntry 4 }

simPsigEvtTrServiceId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event trigger service identifier. "
    ::= { simPsigEvtTrEntry 5 }

simPsigEvtTrEventId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event Identifier. "
    ::= { simPsigEvtTrEntry 6 }

simPsigEvtTrStartTime OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..4))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event start time. "
    ::= { simPsigEvtTrEntry 7 }

```

```

simPsigEvtTrDuration OBJECT-TYPE
    SYNTAX OCTET STRING(SIZE (0..2))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event duration. "
    ::= { simPsigEvtTrEntry 8 }

simPsigEvtTrPrivateData OBJECT-TYPE
    SYNTAX OCTET STRING(SIZE (0..256))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "EVNT Channel Identifier. "
    ::= { simPsigEvtTrEntry 9 }

--
--

simPsigDescInsTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigDescInsEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the information related to descriptor insertion."
    ::= { simPSI 6 }

simPsigDescInsEntry OBJECT-TYPE
    SYNTAX SimPsigDescInsEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigDescInsIndex }
    ::= { simPsigDescInsTable 1 }

SimPsigDescInsEntry ::= SEQUENCE {
    simPsigDescInsIndex          INTEGER (0..65535),
    simPsigDescInsAdminState     AdministrativeState,
    simPsigDescInsTrIndex        INTEGER (0..65535),
    simPsigDescInsTrType         TriggerType,
    simPsigDescInsLocationId     InsertLocation,
    simPsigDescInsNetworkId      INTEGER (0..65535),
    simPsigDescInsONetworkId     INTEGER (0..65535),
    simPsigDescInsTransStreamId  INTEGER (0..65535),
    simPsigDescInsServiceId      INTEGER (0..65535),
    simPsigDescInsElmStreamId    INTEGER (0..65535),
    simPsigDescInsBouquetId      INTEGER (0..65535),
    simPsigDescInsEventId        INTEGER (0..65535),
    simPsigDescInsONetworkId2loop INTEGER (0..65535),
    simPsigDescInsNetworkIdOther INTEGER (0..65535),
    simPsigDescInsTransStreamId2orO INTEGER (0..65535),
    simPsigDescInsDelayType      DelayType,
    simPsigDescInsDelay          OCTET STRING (SIZE(0..1)),
    simPsigDescPrivDataSpfier    INTEGER (0..65535),
    simPsigDescInsEntryStatus    RowStatus
}

simPsigDescInsIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The unique index into the table."
    ::= { simPsigDescInsEntry 1 }

simPsigDescInsAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used by an authorized manager to lock a conceptual row for exclusive
        write and create access."
    ::= { simPsigDescInsEntry 2 }

```

```

simPsigDescInsTrIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The unique index into the corresponding trigger table."
    ::= { simPsigDescInsEntry 3 }

simPsigDescInsTrType OBJECT-TYPE
    SYNTAX TriggerType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The type of the trigger that caused this descriptor insert."
    ::= { simPsigDescInsEntry 4 }

simPsigDescInsLocationId OBJECT-TYPE
    SYNTAX InsertLocation
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The type of target table for insertion."
    ::= { simPsigDescInsEntry 5 }

simPsigDescInsNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier."
    ::= { simPsigDescInsEntry 6 }

simPsigDescInsONetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier."
    ::= { simPsigDescInsEntry 7 }

simPsigDescInsTransStreamId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier. "
    ::= { simPsigDescInsEntry 8 }

simPsigDescInsServiceId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Service Identifier. "
    ::= { simPsigDescInsEntry 9 }

simPsigDescInsElmStreamId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Elementary stream identifier. "
    ::= { simPsigDescInsEntry 10 }

simPsigDescInsBouquetId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event trigger bouquet identifier. "
    ::= { simPsigDescInsEntry 11 }

```

```

simPsigDescInsEventId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "EVNT Identifier. "
    ::= { simPsigDescInsEntry 12 }

simPsigDescInsONetworkId2loop OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier second loop."
    ::= { simPsigDescInsEntry 13 }

simPsigDescInsNetworkIdOther OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier other."
    ::= { simPsigDescInsEntry 14 }

simPsigDescInsTransStreamId2OrO OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier second loop or other. "
    ::= { simPsigDescInsEntry 15 }

simPsigDescInsDelayType OBJECT-TYPE
    SYNTAX DelayType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Delay type, immediate or synchronized. "
    ::= { simPsigDescInsEntry 16 }

simPsigDescInsDelay OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..1))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Insert delay"
    ::= { simPsigDescInsEntry 17 }

simPsigDescPrivDataSpfier OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Private data specifier"
    ::= { simPsigDescInsEntry 18 }

simPsigDescInsEntryStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Other transport stream identifiers."
    ::= { simPsigDescInsEntry 19 }

--
--

simPsigDescInsDescTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigDescInsDescEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the descriptors to be inserted."
    ::= { simPSI 7 }

```

```

simPsigDescInsDescEntry OBJECT-TYPE
    SYNTAX SimPsigDescInsDescEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simPsigDescInsIndex, simPsigDescInsDescIndex }
    ::= { simPsigDescInsDescTable 1 }

SimPsigDescInsDescEntry ::= SEQUENCE {
    simPsigDescInsDescIndex      INTEGER (0..65535),
    simPsigDescInsDescAdminState AdministrativeState,
    simPsigDescInsDescriptor     OCTET STRING (SIZE(0..8191)),
    simPsigDescInsDescriptorStatus DescriptorStatus,
    simPsigDescInsDescEntryStatus RowStatus
}

simPsigDescInsDescIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The unique index into the table."
    ::= { simPsigDescInsDescEntry 1 }

simPsigDescInsDescAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Used by an authorized manager to lock a conceptual row for exclusive
        write and create access."
    ::= { simPsigDescInsDescEntry 2 }

simPsigDescInsDescriptor OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..8191))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The descriptor to be inserted."
    ::= { simPsigDescInsDescEntry 3 }

simPsigDescInsDescriptorStatus OBJECT-TYPE
    SYNTAX DescriptorStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The insertion status of the descriptor."
    ::= { simPsigDescInsDescEntry 4 }

simPsigDescInsDescEntryStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Other transport stream identifiers."
    ::= { simPsigDescInsDescEntry 5 }

--
--

simPsigTblProvTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigTblProvEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table is the interface to obtaining all PSI/SI information."
    ::= { simPSI 8 }

simPsigTblProvEntry OBJECT-TYPE
    SYNTAX SimPsigTblProvEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry."
    INDEX { simPsigTblProvIndex }
    ::= { simPsigTblProvTable 1 }

```

```

SimPsigTblProvEntry ::= SEQUENCE {
    simPsigTblProvIndex      INTEGER (0..65535),
    simPsigTblProvTableId    ProvTableId,
    simPsigTblNetworkId      INTEGER (0..65535),
    simPsigTblONetworkId     INTEGER (0..65535),
    simPsigTblTransStreamId  INTEGER (0..65535),
    simPsigTblServiceId      INTEGER (0..65535),
    simPsigTblBouquetId      INTEGER (0..65535),
    simPsigTblEventId        INTEGER (0..65535),
    simPsigTblONetworkId2loop INTEGER (0..65535),
    simPsigTblNetworkIdOther INTEGER (0..65535),
    simPsigTblTransStreamId2OrO INTEGER (0..65535),
    simPsigTblSegmentNr      INTEGER (0..65535),
    simPsigTblProvPart       OCTET STRING (SIZE(0..8191)),
    simPsigTblProvPartNumber  INTEGER (0..65535)
}

```

```

simPsigTblProvIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The unique index into the table."
    ::= { simPsigTblProvEntry 1 }

```

```

simPsigTblProvTableId OBJECT-TYPE
    SYNTAX ProvTableId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The table identifier of the table."
    ::= { simPsigTblProvEntry 2 }

```

```

simPsigTblNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier."
    ::= { simPsigTblProvEntry 3 }

```

```

simPsigTblONetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier."
    ::= { simPsigTblProvEntry 4 }

```

```

simPsigTblTransStreamId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier. "
    ::= { simPsigTblProvEntry 5 }

```

```

simPsigTblServiceId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Service Identifier. "
    ::= { simPsigTblProvEntry 6 }

```

```

simPsigTblBouquetId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event trigger bouquet identifier. "
    ::= { simPsigTblProvEntry 7 }

```



```

simPsigTblEventId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event Identifier "
    ::= { simPsigTblProvEntry 8 }

simPsigTblONetworkId2loop OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier second loop."
    ::= { simPsigTblProvEntry 9 }

simPsigTblNetworkIdOther OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier other."
    ::= { simPsigTblProvEntry 10 }

simPsigTblTransStreamId2OrO OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier second loop or other. "
    ::= { simPsigTblProvEntry 11 }

simPsigTblSegmentNr OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Segment Number "
    ::= { simPsigTblProvEntry 12 }

simPsigTblProvPart OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..8191))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Event Identifier "
    ::= { simPsigTblProvEntry 13 }

simPsigTblProvPartNumber OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Each table is subdivided into parts for SNMP transport if necessary.
        The part number identifies the table part of this entry. "
    ::= { simPsigTblProvEntry 14 }

--
--

simPsigPIDProvTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigPIDProvEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This is the PID provisioning table."
    ::= { simPSI 9 }

simPsigPIDProvEntry OBJECT-TYPE
    SYNTAX SimPsigPIDProvEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigPIDProvSuCasId, simPsigPIDProvFlowId }
    ::= { simPsigPIDProvTable 1 }

```

```

SimPsigPIDProvEntry ::= SEQUENCE {
    simPsigPIDProvFlowType FlowType,
    simPsigPIDProvSuCasId SuperCasId,
    simPsigPIDProvFlowId FlowId,
    simPsigPIDProvFlowPID INTEGER (0..65535)
}

simPsigPIDProvFlowType OBJECT-TYPE
    SYNTAX FlowType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Flow Type. "
    ::= { simPsigPIDProvEntry 1 }

simPsigPIDProvSuCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Flow Super CAS identifier. "
    ::= { simPsigPIDProvEntry 2 }

simPsigPIDProvFlowId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Flow Stream Identifier. "
    ::= { simPsigPIDProvEntry 3 }

simPsigPIDProvFlowPID OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Flow PID "
    ::= { simPsigPIDProvEntry 4 }

--
--

simPsigPdTrTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SimPsigPdTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "This table contains all the active Private Data Triggers."
    ::= { simPSI 10 }

simPsigPdTrEntry OBJECT-TYPE
    SYNTAX SimPsigPdTrEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single table entry. "
    INDEX { simPsigPdTrIndex }
    ::= { simPsigPdTrTable 1 }

SimPsigPdTrEntry ::= SEQUENCE {
    simPsigPdTrIndex INTEGER (0..65535),
    simPsigPdTrNetworkId INTEGER (0..65535),
    simPsigPdTrOnNetworkId INTEGER (0..65535),
    simPsigPdTrTransStreamId INTEGER (0..65535),
    simPsigPdTrServiceId INTEGER (0..65535),
    simPsigPdTrEsId INTEGER (0..65535),
    simPsigPdTrType ECMTriggerType,
    simPsigPdTrSuCasId SuperCasId,
    simPsigPdTrPdId FlowId,
    simPsigPdTrPdPid INTEGER (0..65535),
    simPsigPdTrPdStreamType INTEGER (0..255),
    simPsigPdTrPrivateData OCTET STRING (SIZE(0..127))
}

```

```

simPsigPdTrIndex OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "The trigger index."
    ::= { simPsigPdTrEntry 1 }

simPsigPdTrNetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Network Identifier."
    ::= { simPsigPdTrEntry 2 }

simPsigPdTrONetworkId OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The Original Network Identifier."
    ::= { simPsigPdTrEntry 3 }

simPsigPdTrTransStreamId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Transport Stream Identifier. "
    ::= { simPsigPdTrEntry 4 }

simPsigPdTrServiceId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Service Identifier. "
    ::= { simPsigPdTrEntry 5 }

simPsigPdTrEsId OBJECT-TYPE
    SYNTAX INTEGER(0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Elementary Stream Identifier. "
    ::= { simPsigPdTrEntry 6 }

simPsigPdTrType OBJECT-TYPE
    SYNTAX ECMTriggerType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Trigger Type. "
    ::= { simPsigPdTrEntry 7 }

simPsigPdTrSuCasId OBJECT-TYPE
    SYNTAX SuperCasId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Client Identifier. "
    ::= { simPsigPdTrEntry 8 }

simPsigPdTrPdId OBJECT-TYPE
    SYNTAX FlowId
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM Stream Identifier. "
    ::= { simPsigPdTrEntry 9 }

```

```

simPsigPdTrPdPid OBJECT-TYPE
    SYNTAX INTEGER (0..65535)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "ECM PID "
    ::= { simPsigPdTrEntry 10 }

simPsigPdTrPdStreamType OBJECT-TYPE
    SYNTAX INTEGER (0..255)
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Stream type of Private Data Stream"
    ::= { simPsigPdTrEntry 11 }

simPsigPdTrPrivateData OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE(0..127))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Private Data string. "
    ::= { simPsigPdTrEntry 12}

-- -----
--
-- Conformance Information
--

simCompliances OBJECT IDENTIFIER ::= { simMIBConformance 1 }
simGroups OBJECT IDENTIFIER ::= { simMIBConformance 2 }

simEcmgCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for SNMP Entities which host or
        represent ECMGs"
    MODULE -- this module
    MANDATORY-GROUPS { simIdentGroup, simEcmgGroup}
    ::= { simCompliances 1}

simEmOrPdCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for SNMP Entities which host or
        represent EMMGs or PDGs"
    MODULE -- this module
    MANDATORY-GROUPS { simIdentGroup, simEmOrPdGroup}
    GROUP simEmOrPdLapGGroup
    DESCRIPTION
        "Allows for grouping of LAPs "
    ::= { simCompliances 2}

simCpsigCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for SNMP Entities which host or
        represent C(P)SIG)s"
    MODULE -- this module
    MANDATORY-GROUPS { simIdentGroup, simCpsigGroup}
    ::= { simCompliances 3}

simPsigCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for SNMP Entities which host or
        represent (P)SIG)s"
    MODULE -- this module
    MANDATORY-GROUPS { simIdentGroup, simPsigGroup}
    ::= { simCompliances 4}

```

```

simIdentGroup OBJECT-GROUP
  OBJECTS {
    simSoftwareVersion,
    simMIBVersion,
    simMIBPrivateVersion,
    simAgentVersion
  }
  STATUS current
  DESCRIPTION
    "A collection of objects providing software configuration information."
    ::= { simGroups 1 }

simEcmgGroup OBJECT-GROUP
  OBJECTS {
    simEcmgIndex,
    simEcmgIpAddress,
    simEcmgTcpPort,
    simEcmgSuCasId,
    simEcmgChannels,
    simEcmgCwPrs,
    simEcmgErrs,
    simEcmgTargetCpsig,
    simEcmgCaMib ,
    simEcmgChannelId,
    simEcmgCSCsIpAddress,
    simEcmgCSCsTcpPort,
    simEcmgCStreams,
    simEcmgCCwPrs,
    simEcmgCErrs,
    simEcmgStreamId,
    simEcmgEcmId,
    simEcmgSLastCp,
    simEcmgSCwPrs,
    simEcmgSErrs
  }
  STATUS current
  DESCRIPTION
    " A collection of objects providing ECMG information."
    ::= { simGroups 2 }

simEmOrPdGroup OBJECT-GROUP
  OBJECTS {
    simEmOrPdIndex,
    simEmOrPdDataType,
    simEmOrPdClientId,
    simEmOrPdCommCapability,
    simEmOrPdErrs,
    simEmOrPdTargetCpsig ,
    simEmOrPdCaMib,
    simEmOrPdLapIndex,
    simEmOrPdLapAdminState,
    simEmOrPdLapCommType,
    simEmOrPdLapMuxIpAddress,
    simEmOrPdLapMuxPort,
    simEmOrPdLapStatus,
    simEmOrPdChannelId,
    simEmOrPdCommType,
    simEmOrPdCIpAddress,
    simEmOrPdCPort,
    simEmOrPdCErrs,
    simEmOrPdDataId,
    simEmOrPdSChannelId,
    simEmOrPdBwidth,
    simEmOrPdStreamId,
    simEmOrPdSErrs,
    simEmOrPdSBytes
  }
  STATUS current
  DESCRIPTION
    " A collection of objects providing EMMG/PDG information."
    ::= { simGroups 3 }

```

```

simEmOrPdLapGGroup OBJECT-GROUP
  OBJECTS {
    simEmOrPdLapGroup,
    simEmOrPdLapGAdminState,
    simEmOrPdLapGStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects providing LAPG infomation."
    ::= { simGroups 4 }

simCpsigGroup OBJECT-GROUP
  OBJECTS {
    simCpsigIndex,
    simCpsigSuperCasId,
    simCpsigErrs,
    simCpsigChannels,
    simCpsigCpsigIpAddress,
    simCpsigCpsigPort,
    simCpsigCaMib,
    simCpsigChannelId,
    simCpsigPsigIpAddress,
    simCpsigPsigPort,
    simCpsigCErrs,
    simCpsigCTstrms,
    simCpsigCSstrms,
    simCpsigStreamTStreamId,
    simCpsigStreamNid,
    simCpsigStreamOnid,
    simCpsigStreamMaxCompTime,
    simCpsigStreamTriggerEnable,
    simCpsigStreamLastTrigger,
    simCpsigStreamLastEventId,
    simCpsigStreamLastServiceId,
    simCpsigStreamLastEsId,
    simCpsigStreamLastEcmPid,
    simCpsigStreamErrs,
    simCpsigStreamBytes
  }
  STATUS current
  DESCRIPTION
    "A collection of objects providing C(P)SIG infomation."
    ::= { simGroups 5 }

simPsigGroup OBJECT-GROUP
  OBJECTS {
    simPsigIndex,
    simPsigType,
    simPsigTriggerSupport,
    simPsigNetworkId,
    simPsigONetworkId,
    simPsigTransStreamId,
    simPsigTSServices,
    simPsigConfigAdminState,
    simPsigConfigCustCasId,
    simPsigConfigMaxCompTime,
    simPsigConfigServiceId,
    simPsigConfigTriggerEnable,
    simPsigConfigEntryStatus,
    simPsigConfigCpsigType,
    simPsigConfigCADInsMode,
    simPsigEcmTrNetworkId,
    simPsigEcmTrONetworkId,
    simPsigEcmTrTransStreamId,
    simPsigEcmTrServiceId,
    simPsigEcmTrEsId,
    simPsigEcmTrType,
    simPsigEcmTrSuCasId,
    simPsigEcmTrEcmId,
    simPsigEcmTrEcmPid,
    simPsigEcmTrAccessCriteria,
    simPsigEvtntTrNetworkId,
    simPsigEvtntTrONetworkId,
    simPsigEvtntTrTransStreamId,
    simPsigEvtntTrServiceId,
    simPsigEvtntTrEventId,
    simPsigEvtntTrStartTime,

```

```

simPsigEvtTrDuration,
simPsigEvtTrPrivateData,
simPsigFlowTrIndex,
simPsigFlowTrType,
simPsigFlowTrSuCasId,
simPsigFlowTrFlowId,
simPsigFlowTrFlowPID,
simPsigDescInsIndex,
simPsigDescInsAdminState,
simPsigDescInsTrIndex,
simPsigDescInsTrType,
simPsigDescInsLocationId,
simPsigDescInsNetworkId,
simPsigDescInsONetworkId,
simPsigDescInsTransStreamId,
simPsigDescInsServiceId,
simPsigDescInsElmStreamId,
simPsigDescInsBouquetId,
simPsigDescInsEventId,
simPsigDescInsNetworkIdOther,
simPsigDescInsONetworkId2loop,
simPsigDescInsTransStreamId2OrO,
simPsigDescInsDelayType,
simPsigDescInsDelay,
simPsigDescPrivDataSpfier,
simPsigDescInsEntryStatus,
simPsigDescInsDescIndex,
simPsigDescInsDescAdminState,
simPsigDescInsDescriptor,
simPsigDescInsDescriptorStatus,
simPsigDescInsDescEntryStatus,
simPsigTblProvIndex,
simPsigTblProvTableId      ,
simPsigTblNetworkId,
simPsigTblONetworkId      ,
simPsigTblTransStreamId    ,
simPsigTblServiceId ,
simPsigTblBouquetId,
simPsigTblEventId      ,
simPsigTblNetworkIdOther,
simPsigTblONetworkId2loop,
simPsigTblTransStreamId2OrO,
simPsigTblSegmentNr,
simPsigTblProvPart,
simPsigTblProvPartNumber,
simPsigPIDProvFlowType,
simPsigPIDProvSuCasId,
simPsigPIDProvFlowId,
simPsigPIDProvFlowPID,
simPsigPdTrNetworkId,
simPsigPdTrONetworkId,
simPsigPdTrTransStreamId,
simPsigPdTrServiceId,
simPsigPdTrEsId,
simPsigPdTrType,
simPsigPdTrSuCasId,
simPsigPdTrPdId,
simPsigPdTrPdPid,
simPsigPdTrPdStreamType,
simPsigPdTrPrivateData
}
STATUS current
DESCRIPTION
    "A collection of objects providing (P)SIG infomation."
    ::= { simGroups 6 }

```

END

K.2 SEM MIB

```
SEM-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
MODULE-IDENTITY, OBJECT-TYPE,
Integer32, Unsigned32, BITS, IPAddress, TimeTicks,
NOTIFICATION-TYPE FROM SNMPv2-SMI
TEXTUAL-CONVENTION, RowStatus,
DisplayString, TruthValue, DateAndTime FROM SNMPv2-TC
MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP FROM SNMPv2-CONF;
```

```
semMIB MODULE-IDENTITY
```

```
LAST-UPDATED "9707021700Z "
ORGANIZATION "DVB Simulcrypt Technical Group "
CONTACT-INFO " --- "
DESCRIPTION
"The MIB module for defining DVB Simulcrypt Conditional
Access System event information. "
::= {1 3 6 1 4 1 2696 1 2}
```

```
EntryName ::= TEXTUAL-CONVENTION
```

```
STATUS current
DESCRIPTION
" Entry name convention for tables. "
SYNTAX OCTET STRING (SIZE (8))
```

```
EventType ::= TEXTUAL-CONVENTION
```

```
STATUS current
DESCRIPTION
"An event type is indicated if the bit corresponding to its power of
two is set, i.e. if the mask is 3 then communications and
qualityOfService event types are indicated. "
SYNTAX BITS {
communications(0),
qualityOfService(1),
processingError(2),
equipmentAlarm(3),
environmental(4),
attributeValueChange(5),
stateChange(6),
timeDomainViolation(7),
securitySrvOrMechnsmViolation(8),
relationshipChange(9),
operationalViolation(10),
integrityViolation(11),
physicalViolation(12),
thresholdCrossing(13),
thresholdClearing(14)
}
```

```
ProbableCause ::= TEXTUAL-CONVENTION
```

```
STATUS current
DESCRIPTION
"Defined in ITU-T Recommendation X.733 [5] "
SYNTAX BITS {
simulcryptSpecific(0),
adapterError(1),
applicationSubsystemFailure(2),
bandwidthReduced(3),
callEstablishmentError(4),
communicationsProtocolError(5),
communicationsSubsystemFailure(6),
configurationOrCustomizationError(7),
congestion(8),
corruptData(9),
cpuCyclesLimitExceeded(10),
dataSetOrModemError(11),
degradedSignal(12),
dTEDCEInterfaceError(13),
enclosureDoorOpen(14),
equipmentMalfunction(15),
excessiveVibration(16),
fileError(17),
fireDetected(18),
floodDetected(19),
```



```

    framingError(20),
    heatOrVentOrCoolSystemProblem(21),
    humidityUnacceptable(22),
    inputOutputDeviceError(23),
    inputDeviceError(24),
    lANError(25),
    leakDetected(26),
    localNodeTransmissionError(27),
    lossOfFrame(28),
    lossOfSignal(29),
    materialSupplyExhausted(30),
    multiplexerProblem(31),
    outOfMemory(32),
    ouputDeviceError(33),
    performanceDegraded(34),
    powerProblem(35),
    pressureUnacceptable(36),
    processorProblem(37),
    pumpFailure(38),
    queueSizeExceeded(39),
    receiveFailure(40),
    receiverFailure(41),
    remoteNodeTransmissionError(42),
    resourceAtOrNearingCapacity(43),
    responseTimeExcessive(44),
    retransmissionRateExcessive(45),
    softwareError(46),
    softProgramAbnormallyTerminated(47),
    softwareProgramError(48),
    storageCapacityProblem(49),
    temperatureUnacceptable(50),
    thresholdCrossed(51),
    timingProblem(52),
    toxicLeakDetected(53),
    transmitFailure(54),
    transmitterFailure(55),
    underlyingResourceUnavailable(56),
    versionMismatch(57),
    authenticationFailure(58),
    breachOfConfidentiality(59),
    cableTamper(60),
    delayedInformation(61),
    denialOfService(62),
    duplicateInformation(63),
    informationMissing(64),
    informationModificationDetected(65),
    informationOutOfSequence(66),
    intrusionDetection(67),
    keyExpired(68),
    nonRepudiationFailure(69),
    outOfHoursActivity(70),
    outOfService(71),
    proceduralError(72),
    unauthorizedAccessAttempt(73),
    unexpectedInformation(74),
    unspecifiedReason(75)
}

```

PerceivedSeverity ::= TEXTUAL-CONVENTION

STATUS current

DESCRIPTION

"This convention defines six severity levels, which provide an indication of how it is perceived that the capability of the managed object has been affected. Those severity levels which represent service affecting conditions ordered from most severe to least severe are Critical, Major, Minor, and Warning.

Perceived Severity is defined in ITU-T Recommendation X.733 [5]. "

SYNTAX BITS

```

{
    cleared(0),
    indeterminate(1),
    warning(2),
    minor(3),
    major(4),
    critical(5)
}

```

```

}

TrendIndication ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the trend of an event as defined in ITU-T Recommendation X.733 [5]. "
    SYNTAX BITS
        {
            lessSevere(0),
            noChange(1),
            moreSevere(2)
        }

BackedUpStatus ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The backed up status as defined in ITU-T Recommendation X.733 [5]. "
    SYNTAX BITS
        {
            backedUp(0),
            notBackedUp(1)
        }

AdministrativeState ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Administrative state as defined by ITU-T Recommendation X.734 [6]. "
    SYNTAX BITS
        {
            locked(0),
            unlocked(1),
            shuttingDown(2)
        }

OperationalState ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Operational state as defined by ITU-T Recommendation X.734 [6]. "
    SYNTAX BITS
        {
            enabled(0),
            disabled(1)
        }

AvailabilityStatus ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Availability Status as defined by ITU-T Recommendation X.734 [6]. "
    SYNTAX BITS
        {
            available(0),
            inTest(1),
            failed(2),
            powerOff(3),
            offLine(4),
            offDuty(5),
            dependency(6),
            degraded(7),
            notInstalled(8),
            logFull(9)
        }

AlarmStatus ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Alarm Status as defined by ITU-T Recommendation X.734 [6]. "
    SYNTAX BITS
        {
            underRepair(0),
            critical(1),
            major(2),
            minor(3),
            alarmOutstanding(4),
            cleared(5)
        }

```

```

EventSensitivity ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Is the event caused by crossing of a threshold (edge) or by exceeding a threshold
        (level). "
    SYNTAX BITS
    {
        edgeSensitive(0),
        levelSensitive(1)
    }

semMIBObjects          OBJECT IDENTIFIER ::= {semMIB 1}
semMIBConformance     OBJECT IDENTIFIER ::= {semMIB 2}
semMIBNotificationPrefix OBJECT IDENTIFIER ::= {semMIB 3}

semEvent OBJECT IDENTIFIER ::= {semMIBObjects 1}
semEfd OBJECT IDENTIFIER ::= {semMIBObjects 2}

--
-- Event Section
--

semEventTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SemEventEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A table of management event information. "
    ::= {semEvent 1}

semEventEntry OBJECT-TYPE
    SYNTAX SemEventEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single management event. "
    INDEX {semEventName}
    ::= {semEventTable 1}

SemEventEntry ::= SEQUENCE {
    semEventName      EntryName,
    semEventAdminState AdministrativeState,
    semEventAlarmStatus AlarmStatus,
    semEventType      EventType,
    semEventText       DisplayString,
    semEventChangedObjectId OBJECT IDENTIFIER,
    semEventToStateChange Unsigned32,
    semEventRisingThreshold Integer32,
    semEventFallingThreshold Integer32,
    semEventProbableCause ProbableCause,
    semEventPerceivedSeverity PerceivedSeverity,
    semEventTrendIndication TrendIndication,
    semEventBackedUpStatus BackedUpStatus,
    semEventBackUpObject OBJECT IDENTIFIER,
    semEventSpecificProblems OBJECT IDENTIFIER,
    semEventFrequency Integer32,
    semEventSensitivity EventSensitivity,
    semEventStatus      RowStatus
}

semEventName OBJECT-TYPE
    SYNTAX EntryName
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The unique name of the target event. "
    ::= {semEventEntry 1}

semEventAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION

```

```

"The Administrative State of the event entry. "
::= {semEventEntry 2}

semEventAlarmStatus OBJECT-TYPE
SYNTAX AlarmStatus
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"The Alarm Status of an event. "
::= {semEventEntry 3}

semEventType OBJECT-TYPE
SYNTAX EventType
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" Indicates the type of the event. "
::= {semEventEntry 4}

semEventText OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"A description of the event's function and use. "
DEFVAL {'H'}
::= {semEventEntry 5}

semEventChangedObjectId OBJECT-TYPE
SYNTAX OBJECT IDENTIFIER
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"The object identifier of the MIB object to check to see
if the event should fire.

This may be wildcarded by truncating all or part of the
instance portion, in which case the condition is obtained
as if with a GetNext function, checking multiple values
if they exist. "
::= {semEventEntry 6}

semEventToStateChange OBJECT-TYPE
SYNTAX Unsigned32
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" If semEvent ChangedObjectId is a state/status/variable,
this variable identifies the state that causes the
event to be generated. "
::= {semEventEntry 7}

semEventRisingThreshold OBJECT-TYPE
SYNTAX Integer32
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" A threshold value to check against if semEventType is
'threshold'. In this case if the value of the object at
semEventValueID is greater than or equal to this threshold
and the value at the last sampling interval was less than
this threshold, one semEventRisingEvent is triggered.
If semEventType is not 'threshold', this object is not
instantiated. "
DEFVAL {0}
::= {semEventEntry 8}

```

```

semEventFallingThreshold OBJECT-TYPE
    SYNTAX Integer32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " A threshold value to check against if semEventType is
        'threshold'. In this case if the value of the object at
        semEventValueID is less than or equal to this threshold
        and the value at the last sampling interval was greater than
        this threshold, one semEventFallingEvent is triggered.
        If semEventType is not 'threshold', this object is not
        instantiated. "
    DEFVAL {0}
    ::= {semEventEntry 9}

semEventProbableCause OBJECT-TYPE
    SYNTAX ProbableCause
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This variable defines further probable cause for
        the last event of this type. "
    ::= {semEventEntry 10}

semEventPerceivedSeverity OBJECT-TYPE
    SYNTAX PerceivedSeverity
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This parameter defines the perceived severity of the
        last event of this type. "
    ::= {semEventEntry 11}

semEventTrendIndication OBJECT-TYPE
    SYNTAX TrendIndication
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates the trend of the last event of this type. "
    ::= {semEventEntry 12}

semEventBackedUpStatus OBJECT-TYPE
    SYNTAX BackedUpStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The backed up status. "
    ::= {semEventEntry 13}

semEventBackUpObject OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " If the backed up status is backedUp then this variable
        contains the object identifier of the object containing
        back up object. "
    ::= {semEventEntry 14}

semEventSpecificProblems OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        " This variable identifies the object responsible for the event. "
    ::= {semEventEntry 15}

```

```

semEventFrequency OBJECT-TYPE
    SYNTAX Integer32 (1..65535)
    UNITS "seconds "
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " The number of seconds to wait between event condition
          checks. To encourage consistency in sampling, the
          interval is measured from the beginning of one check to
          the beginning of the next. "
    DEFVAL {600}
    ::= {semEventEntry 16}

semEventSensitivity OBJECT-TYPE
    SYNTAX EventSensitivity
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The event sensitivity which identifies whether the event is level or edge
          sensitive. "
    ::= {semEventEntry 17}

semEventStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The control that allows creation/deletion of entries. "
    ::= {semEventEntry 18}

--
-- Event Forwarding Discriminator (EFD) Status Section
--

semEfdTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SemEfdEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A table of management EFDs. "
    ::= {semEfd 1}

semEfdEntry OBJECT-TYPE
    SYNTAX SemEfdEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "Information about a single EFD. "
    INDEX {semEfdName , semEfdTarget}
    ::= {semEfdTable 1}

SemEfdEntry ::= SEQUENCE {
    semEfdName      EntryName,
    semEfdAdminState      AdministrativeState,
    semEfdOperState      OperationalState,
    semEfdAvailStatus      AvailabilityStatus,
    semEfdStartTime      DateAndTime,
    semEfdStopTime      DateAndTime,
    semEfdDailyStartTime      TimeTicks,
    semEfdDailyStopTime      TimeTicks,
    semEfdWeeklyMask      OCTET STRING,
    semEfdTypes      EventType,
    semEfdCause      ProbableCause,
    semEfdSeverity      PerceivedSeverity,
    semEfdSpecificProblems      OBJECT IDENTIFIER,
    semEfdTrendIndication      TrendIndication,
    semEfdChangedObjectId      OBJECT IDENTIFIER,
    semEfdToStateChange      Unsigned32,
    semEfdNotification      OBJECT IDENTIFIER,
    semEfdOr      TruthValue,
    semEfdTarget      IpAddress,
    semEfdText      DisplayString,
    semEfdStatus      RowStatus
}

```

```

semEfdName OBJECT-TYPE
    SYNTAX EntryName
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The unique name of the EFD. "
    ::= {semEfdEntry 1}

semEfdAdminState OBJECT-TYPE
    SYNTAX AdministrativeState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " The current Administrative state of the EFD. "
    ::= {semEfdEntry 2}

semEfdOperState OBJECT-TYPE
    SYNTAX OperationalState
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " The current Operational state of the EFD. "
    ::= {semEfdEntry 3}

semEfdAvailStatus OBJECT-TYPE
    SYNTAX AvailabilityStatus
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        " This object controls the Availability status of the EFD
          which reflects the scheduling. "
    ::= {semEfdEntry 4}

semEfdStartTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the date and time at which an unlocked and
          enabled EFD starts functioning, i.e. changes its
          availability status from offDuty to available. "
    ::= {semEfdEntry 5}

semEfdStopTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the date and time at which an unlocked and
          enabled EFD stops functioning, i.e. changes its
          availability status from available to offDuty. "
    ::= {semEfdEntry 6}

semEfdDailyStartTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the daily start time at which an unlocked and
          enabled EFD starts functioning, i.e. changes its
          availability status from offDuty to available. "
    ::= {semEfdEntry 7}

semEfdDailyStopTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the daily stop time at which an unlocked and
          enabled EFD stops functioning, i.e. changes its
          availability status from available to offDuty. "
    ::= {semEfdEntry 8}

```

```

semEfdWeeklyMask OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (1))
    MAX-ACCESS read-create
STATUS current
DESCRIPTION
    " This variable defines the weekly schedule at which an unlocked and
      enabled EFD may start functioning, i.e. changes its
      availability status from available to offDuty. A day is
      scheduled if the corresponding power of 2, i.e. 2**3 for
      Wednesday is in the mask. "
 ::= {semEfdEntry 9}

semEfdTypes OBJECT-TYPE
    SYNTAX EventType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " The event types that this EFD may generate notifications for. "
 ::= {semEfdEntry 10}

semEfdCause OBJECT-TYPE
    SYNTAX ProbableCause
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " Any event with a different probable cause is ignored. "
 ::= {semEfdEntry 11}

semEfdSeverity OBJECT-TYPE
    SYNTAX PerceivedSeverity
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " Any event with severity equal to or less the discriminated level
      is ignored. "
 ::= {semEfdEntry 12}

semEfdSpecificProblems OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " Any event not generated by the identified object is ignored. "
 ::= {semEfdEntry 13}

semEfdTrendIndication OBJECT-TYPE
    SYNTAX TrendIndication
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " Any events with a trend less or equal to specified value are
      ignored. "
 ::= {semEfdEntry 14}

semEfdChangedObjectId OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    "Any events not caused by a change of the value of this object
      are ignored. "
 ::= {semEfdEntry 15}

semEfdToStateChange OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
    " Any state changes to a different state than the one indicated are
      ignored. "
 ::= {semEfdEntry 16}

```



```

semEfdNotification OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The object identifier from the NOTIFICATION-TYPE for the
        notification. "
    ::= {semEfdEntry 17}

semEfdOr OBJECT-TYPE
    SYNTAX TruthValue
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Indicates whether if this EFD matches the event, the matching
        process shall be continued with the next EFD. "
    ::= {semEfdEntry 18}

semEfdTarget OBJECT-TYPE
    SYNTAX IpAddress
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "Notifications Targets. A value of 0
        indicates the local system. "
    ::= {semEfdEntry 19}

semEfdText OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "A description of the EFD's function and use. "
    ::= {semEfdEntry 20}

semEfdStatus OBJECT-TYPE
    SYNTAX RowStatus
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        "The control that allows creation/deletion of entries. "
    ::= {semEfdEntry 21}

--
-- Notifications
--

semMIBNotifications OBJECT IDENTIFIER ::= {semMIBNotificationPrefix 0}

semEventAlarm NOTIFICATION-TYPE
    OBJECTS
        {
            semEventName,
            semEventType,
            semEventProbableCause,
            semEventSpecificProblems,
            semEventPerceivedSeverity,
            semEventTrendIndication,
            semEventText
        }
    STATUS current
    DESCRIPTION
        " Alarm notification. "
    ::= {semMIBNotifications 1}

```

```

semEventStateChange NOTIFICATION-TYPE
  OBJECTS
    {
      semEventName,
      semEventToStateChange,
      semEventChangedObjectId
    }
  STATUS current
  DESCRIPTION
    " State change notification. "
  ::= {semMIBNotifications 2}

semEventObjectValueChange NOTIFICATION-TYPE
  OBJECTS
    {
      semEventName,
      semEventChangedObjectId
    }
  STATUS current
  DESCRIPTION
    " Object value change. "
  ::= {semMIBNotifications 3}

--
-- Conformance Information
--

semCompliances      OBJECT IDENTIFIER ::= {semMIBConformance 1}
semGroups           OBJECT IDENTIFIER ::= {semMIBConformance 2}

semECMGCompliance  MODULE-COMPLIANCE
  STATUS current
  DESCRIPTION
    " The compliance statement for SNMP Entities which host or
    represent ECMGs. A hosting entity must also support either
    threshold, state change, or value change events. "
  MODULE -- this module
  MANDATORY-GROUPS {semMandatoryNotifications}
  GROUP semThresholdEventGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEventOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semThresholdEfdGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEfdOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semStateChangeEventGroup
  DESCRIPTION
    " This Group is required if stateChange events are supported. "
  GROUP semStateChangeEventOptGroup
  DESCRIPTION
    " This Group is optional if stateChange events are supported. "
  GROUP semStateChangeEfdGroup
  DESCRIPTION
    " This Group is required if stateChange events are supported. "
  GROUP semStateChangeEfdOptGroup
  DESCRIPTION
    " This Group is optional if stateChange events are supported. "
  GROUP semValueChangeEventGroup
  DESCRIPTION
    " This Group is required valueChange events are supported. "
  GROUP semValueChangeEventOptGroup
  DESCRIPTION
    " This Group is optional valueChange events are supported. "
  GROUP semValueChangeEfdGroup
  DESCRIPTION
    " This Group is required valueChange events are supported. "
  GROUP semValueChangeEfdOptGroup
  DESCRIPTION
    " This Group is optional valueChange events are supported. "
  GROUP semOptionalNotifications
  DESCRIPTION
    " This Group is required if state change or value change notifications
    are supported. "
  ::= {semCompliances 1}

```

```

semEmOrPdCompliance MODULE-COMPLIANCE
  STATUS current
  DESCRIPTION
    " The compliance statement for SNMP Entities which host or
    represent EMMG/PDGs. A hosting entity must also support either
    threshold, state change, or value change events. "
  MODULE -- this module
  MANDATORY-GROUPS {semMandatoryNotifications}
  GROUP semThresholdEventGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEventOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semThresholdEfdGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEfdOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semStateChangeEventGroup
  DESCRIPTION
    " This Group is required if stateChange events are supported. "
  GROUP semStateChangeEventOptGroup
  DESCRIPTION
    " This Group is optional if stateChange events are supported. "
  GROUP semStateChangeEfdGroup
  DESCRIPTION
    " This Group is required if stateChange events are supported. "
  GROUP semStateChangeEfdOptGroup
  DESCRIPTION
    " This Group is optional if stateChange events are supported. "
  GROUP semValueChangeEventGroup
  DESCRIPTION
    " This Group is required valueChange events are supported. "
  GROUP semValueChangeEventOptGroup
  DESCRIPTION
    " This Group is optional valueChange events are supported. "
  GROUP semValueChangeEfdGroup
  DESCRIPTION
    " This Group is required valueChange events are supported. "
  GROUP semValueChangeEfdOptGroup
  DESCRIPTION
    " This Group is optional valueChange events are supported. "
  GROUP semOptionalNotifications
  DESCRIPTION
    " This Group is required if state change or value change notifications
    are supported. "
  ::= {semCompliances 2}

semCpsigCompliance MODULE-COMPLIANCE
  STATUS current
  DESCRIPTION
    " The compliance statement for SNMP Entities which host or
    represent CPSIGs. A hosting entity must also support either
    threshold, state change, or value change events. "
  MODULE -- this module
  MANDATORY-GROUPS {semMandatoryNotifications}
  GROUP semThresholdEventGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEventOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semThresholdEfdGroup
  DESCRIPTION
    " This group is required if threshold events are supported. "
  GROUP semThresholdEfdOptGroup
  DESCRIPTION
    " This group is optional if threshold events are supported. "
  GROUP semStateChangeEventGroup
  DESCRIPTION
    " This Group is required if stateChange events are supported. "
  GROUP semStateChangeEventOptGroup
  DESCRIPTION
    " This Group is optional if stateChange events are supported. "
  GROUP semStateChangeEfdGroup

```

```

DESCRIPTION
  " This Group is required if stateChange events are supported. "
GROUP semStateChangeEfdOptGroup
DESCRIPTION
  " This Group is optional if stateChange events are supported. "
GROUP semValueChangeEventGroup
DESCRIPTION
  " This Group is required valueChange events are supported. "
GROUP semValueChangeEventOptGroup
DESCRIPTION
  " This Group is optional valueChange events are supported. "
GROUP semValueChangeEfdGroup
DESCRIPTION
  " This Group is required valueChange events are supported. "
GROUP semValueChangeEfdOptGroup
DESCRIPTION
  " This Group is optional valueChange events are supported. "
GROUP semOptionalNotifications
DESCRIPTION
  " This Group is required if state change or value change notifications
are supported. "
::= {semCompliances 3}

semCsigCompliance  MODULE-COMPLIANCE
STATUS current
DESCRIPTION
  " The compliance statement for SNMP Entities which host or
represent CSIGs. A hosting entity must also support either
threshold, state change, or value change events. "
MODULE -- this module
MANDATORY-GROUPS {semMandatoryNotifications}
GROUP semThresholdEventGroup
DESCRIPTION
  " This group is required if threshold events are supported. "
GROUP semThresholdEventOptGroup
DESCRIPTION
  " This group is optional if thresold events are supported. "
GROUP semThresholdEfdGroup
DESCRIPTION
  " This group is required if threshold events are supported. "
GROUP semThresholdEfdOptGroup
DESCRIPTION
  " This group is optional if threshold events are supported. "
GROUP semStateChangeEventGroup
DESCRIPTION
  " This Group is required if stateChange events are supported. "
GROUP semStateChangeEventOptGroup
DESCRIPTION
  " This Group is optional if stateChange events are supported. "
GROUP semStateChangeEfdGroup
DESCRIPTION
  " This Group is required if stateChange events are supported. "
GROUP semStateChangeEfdOptGroup
DESCRIPTION
  " This Group is optional if stateChange events are supported. "
GROUP semValueChangeEventGroup
DESCRIPTION
  " This Group is required valueChange events are supported. "
GROUP semValueChangeEventOptGroup
DESCRIPTION
  " This Group is optional valueChange events are supported. "
GROUP semValueChangeEfdGroup
DESCRIPTION
  " This Group is required valueChange events are supported. "
GROUP semValueChangeEfdOptGroup
DESCRIPTION
  " This Group is optional valueChange events are supported. "
GROUP semOptionalNotifications
DESCRIPTION
  " This Group is required if state change or value change notifications
are supported. "
::= {semCompliances 4}

```

```

semMandatoryNotifications    NOTIFICATION-GROUP
  NOTIFICATIONS {
    semEventAlarm
  }
  STATUS current
  DESCRIPTION
    " A collection of objects defining mandatory notifications. "
    ::= {semGroups 1}

semOptionalNotifications     NOTIFICATION-GROUP
  NOTIFICATIONS {
    semEventStateChange,
    semEventObjectValueChange
  }
  STATUS current
  DESCRIPTION
    " A collection of objects defining optional notifications. "
    ::= {semGroups 2}

semThresholdEventGroup       OBJECT-GROUP
  OBJECTS {
    semEventName,
    semEventAdminState,
    semEventType,
    semEventText,
    semEventChangedObjectId,
    semEventRisingThreshold,
    semEventFallingThreshold,
    semEventProbableCause,
    semEventPerceivedSeverity,
    semEventTrendIndication,
    semEventFrequency,
    semEventStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying threshold events. "
    ::= {semGroups 3}

semThresholdEventOptGroup    OBJECT-GROUP
  OBJECTS {
    semEventAlarmStatus,
    semEventBackedUpStatus,
    semEventBackUpObject,
    semEventSpecificProblems,
    semEventSensitivity
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying threshold events. "
    ::= {semGroups 4}

semStateChangeEventGroup     OBJECT-GROUP
  OBJECTS {
    semEventName,
    semEventAdminState,
    semEventType,
    semEventText,
    semEventChangedObjectId,
    semEventToStateChange,
    semEventFrequency,
    semEventStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying state change events. "
    ::= {semGroups 5}

```

```

semStateChangeEventOptGroup OBJECT-GROUP
  OBJECTS {
    semEventAlarmStatus,
    semEventProbableCause,
    semEventPerceivedSeverity,
    semEventTrendIndication,
    semEventBackedUpStatus,
    semEventBackUpObject,
    semEventSpecificProblems,
    semEventSensitivity
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying state change events. "
    ::= {semGroups 6}

semValueChangeEventGroup OBJECT-GROUP
  OBJECTS {
    semEventName,
    semEventAdminState,
    semEventType,
    semEventText,
    semEventChangedObjectId,
    semEventFrequency,
    semEventStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying value change events. "
    ::= {semGroups 7}

semValueChangeEventOptGroup OBJECT-GROUP
  OBJECTS {
    semEventAlarmStatus,
    semEventProbableCause,
    semEventPerceivedSeverity,
    semEventTrendIndication,
    semEventBackedUpStatus,
    semEventBackUpObject,
    semEventSpecificProblems,
    semEventSensitivity
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying value change events. "
    ::= {semGroups 8}

semThresholdEfdGroup OBJECT-GROUP
  OBJECTS {
    semEfdName,
    semEfdAdminState,
    semEfdOperState,
    semEfdAvailStatus,
    semEfdTypes,
    semEfdCause ,
    semEfdSeverity,
    semEfdTrendIndication ,
    semEfdChangedObjectId,
    semEfdNotification,
    semEfdOr,
    semEfdTarget,
    semEfdText,
    semEfdStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying threshold EFDs. "
    ::= {semGroups 9}

```

```

semThresholdEfdOptGroup OBJECT-GROUP
  OBJECTS {
    semEfdStartTime,
    semEfdStopTime,
    semEfdDailyStartTime,
    semEfdDailyStopTime,
    semEfdWeeklyMask,
    semEfdSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying threshold EFDs. "
    ::= {semGroups 10}

semStateChangeEfdGroup OBJECT-GROUP
  OBJECTS {
    semEfdName,
    semEfdAdminState,
    semEfdOperState,
    semEfdAvailStatus,
    semEfdTypes,
    semEfdToStateChange,
    semEfdNotification,
    semEfdOr,
    semEfdTarget,
    semEfdText,
    semEfdStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying state change EFDs. "
    ::= {semGroups 11}

semStateChangeEfdOptGroup OBJECT-GROUP
  OBJECTS {
    semEfdStartTime,
    semEfdStopTime,
    semEfdDailyStartTime,
    semEfdDailyStopTime,
    semEfdCause ,
    semEfdSeverity,
    semEfdTrendIndication ,
    semEfdChangedObjectId,
    semEfdWeeklyMask,
    semEfdSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying state change EFDs. "
    ::= {semGroups 12}

semValueChangeEfdGroup OBJECT-GROUP
  OBJECTS {
    semEfdName,
    semEfdAdminState,
    semEfdOperState,
    semEfdAvailStatus,
    semEfdTypes,
    semEfdChangedObjectId,
    semEfdNotification,
    semEfdOr,
    semEfdTarget,
    semEfdText,
    semEfdStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying value change EFDs. "
    ::= {semGroups 13}

```

```

semValueChangeEfdOptGroup    OBJECT-GROUP
    OBJECTS {
        semEfdStartTime,
        semEfdStopTime,
        semEfdDailyStartTime,
        semEfdDailyStopTime,
        semEfdCause ,
        semEfdSeverity,
        semEfdTrendIndication ,
        semEfdWeeklyMask,
        semEfdSpecificProblems
    }
    STATUS current
    DESCRIPTION
        " A collection of optional objects specifying value change EFDs. "
        ::= {semGroups 14}
END

```

K.3 SLM MIB

SLM-MIB DEFINITIONS ::= BEGIN

```

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Unsigned32, BITS, TimeTicks FROM SNMPv2-SMI
    RowStatus, DisplayString, DateAndTime FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP FROM SNMPv2-CONF
    EventType, ProbableCause, PerceivedSeverity, TrendIndication,
    AdministrativeState, OperationalState,
    AvailabilityStatus, EntryName FROM SEM-MIB;

```

```

slmMIB MODULE-IDENTITY
    LAST-UPDATED "9708071700Z "
    ORGANIZATION "DVB Simulcrypt Technical Group "
    CONTACT-INFO " --- "
    DESCRIPTION
        "The MIB module for defining DVB Simulcrypt Conditional
        Access System logs information. "
        ::= {1 3 6 1 4 1 2696 1 3}

```

```

slmMIBObjects      OBJECT IDENTIFIER ::= {slmMIB 1}
slmMIBConformance  OBJECT IDENTIFIER ::= {slmMIB 2}

```

```

slmLogControl OBJECT IDENTIFIER ::= {slmMIBObjects 1}
slmLogs OBJECT IDENTIFIER ::= {slmMIBObjects 2}

```

```

--
-- Log Control Group
--

```

```

slmLogDefinitionTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SlmLogDefinitionEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A list of Log Table Entry Definitions. Identifies log table
        and the types of events to be logged into that table. "
    REFERENCE " -- "
    ::= {slmLogControl 1}

```

```

slmLogDefinitionEntry OBJECT-TYPE
    SYNTAX SlmLogDefinitionEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "An entry (conceptual row) in the Log Definition Table. "
    REFERENCE " -- "
        INDEX {slmLogDefinitionName }
    ::= {slmLogDefinitionTable 1}

```



```

slmLogDefinitionEntry ::= SEQUENCE
{
    slmLogDefinitionName      EntryName,
    slmLogDefinitionId        OBJECT IDENTIFIER,
    slmLogDefinitionAdminState AdministrativeState,
    slmLogDefinitionOperState OperationalState,
    slmLogDefinitionAvailStatus AvailabilityStatus,
    slmLogDefinitionFullAction BITS,
    slmLogDefinitionMaxLogSize INTEGER,
    slmLogDefinitionCurrentLogSize INTEGER,
    slmLogDefinitionNumberOfRecords INTEGER
}

slmLogDefinitionName OBJECT-TYPE
SYNTAX EntryName
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" The variable used for identifying log Definition table entries. "
REFERENCE " -- "
::= {slmLogDefinitionEntry 1}

slmLogDefinitionId OBJECT-TYPE
SYNTAX OBJECT IDENTIFIER
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" The variable used for identifying log tables. "
REFERENCE " -- "
::= {slmLogDefinitionEntry 2}

slmLogDefinitionAdminState OBJECT-TYPE
SYNTAX AdministrativeState
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" The current Adminsitrative state of the LOG as defined in
   ISO/IEC 10164-2 [4] "
::= {slmLogDefinitionEntry 3}

slmLogDefinitionOperState OBJECT-TYPE
SYNTAX OperationalState
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" The current Operational state of the LOG as defined in
   ISO/IEC 10164-2 [4] "
::= {slmLogDefinitionEntry 4}

slmLogDefinitionAvailStatus OBJECT-TYPE
SYNTAX AvailabilityStatus
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" This object Definitions the Availability status of the LOG
   as defined in ISO/IEC 10164-2 [4] and state-machine "
::= {slmLogDefinitionEntry 5}

slmLogDefinitionFullAction OBJECT-TYPE
SYNTAX BITS
{
    wrap(0),
    halt(1)
}
MAX-ACCESS read-create
STATUS current
DESCRIPTION
" This object Definitions the action to be taken when the maximum
   size of the log has been reached. "
::= {slmLogDefinitionEntry 6}

```

```

slmLogDefinitionMaxLogSize OBJECT-TYPE
SYNTAX INTEGER (1..4294967295)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" This object specifies the maximum size of a log in number
  of octets. A size of 2**32 - 1 specifies that there is no limit. "
 ::= {slmLogDefinitionEntry 7}

slmLogDefinitionCurrentLogSize OBJECT-TYPE
SYNTAX INTEGER (1..4294967295)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" This object specifies the current size of a log in number
  of octets. "
 ::= {slmLogDefinitionEntry 8}

slmLogDefinitionNumberOfRecords OBJECT-TYPE
SYNTAX INTEGER (1..4294967295)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" This object specifies the number of records in the log. "
REFERENCE "ITU-T Recommendation X.735 [7] "
 ::= {slmLogDefinitionEntry 9}

slmLogControlTable OBJECT-TYPE
SYNTAX SEQUENCE OF SlmLogControlEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"A list of Log Table Filter Definitions. Since
  multiple types of events can be logged into the same table,
  multiple entries in the log table could be defining the same
  log table. "
REFERENCE " -- "
 ::= {slmLogControl 2}

slmLogControlEntry OBJECT-TYPE
SYNTAX SlmLogControlEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"An entry (conceptual row) in the Log Control Table. "
REFERENCE " -- "
INDEX      {slmLogDefinitionName, slmLogControlName }
 ::= {slmLogControlTable 1}

SlmLogControlEntry ::= SEQUENCE
{
    slmLogControlName      EntryName,
    slmLogControlStartTime DateAndTime,
    slmLogControlStopTime  DateAndTime,
    slmLogControlDailyStartTime TimeTicks,
    slmLogControlDailyStopTime TimeTicks,
    slmLogControlWeeklyMask OCTET STRING,
    slmLogControlTypes      EventType,
    slmLogControlCause       ProbableCause,
    slmLogControlSeverity    PerceivedSeverity,
    slmLogControlSpecificProblems OBJECT IDENTIFIER,
    slmLogControlToStateChange Unsigned32,
    slmLogControlTrendIndication TrendIndication,
    slmLogControlChangedObjectId OBJECT IDENTIFIER,
    slmLogControlStatus      RowStatus
}

slmLogControlName OBJECT-TYPE
SYNTAX EntryName
MAX-ACCESS read-only
STATUS current
DESCRIPTION
" The variable used for identifying log control table entries. "
REFERENCE " -- "
 ::= {slmLogControlEntry 1}

```

```

slmLogControlStartTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the date and time at which an unlocked and
          enabled log control entry starts functioning, i.e. changes its
          availability status from offDuty to available. "
    ::= {slmLogControlEntry 2}

slmLogControlStopTime OBJECT-TYPE
    SYNTAX DateAndTime
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the date and time at which an unlocked and
          enabled log control stops functioning, i.e. changes its
          availability status from available to offDuty. "
    ::= {slmLogControlEntry 3}

slmLogControlDailyStartTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the daily start time at which an unlocked and
          enabled log control entry starts functioning, i.e. changes its
          availability status from offDuty to available. "
    ::= {slmLogControlEntry 4}

slmLogControlDailyStopTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the daily start time at which an unlocked and
          enabled log control entry stops functioning, i.e. changes its
          availability status from available to offDuty. "
    ::= {slmLogControlEntry 5}

slmLogControlWeeklyMask OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (1))
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the weekly schedule at which an unlocked and
          enabled log control entry may start functioning, i.e. changes its
          availability status from available to offDuty. A day is
          scheduled if the corresponding power of 2, i.e. 2**3 for
          Wednesday is in the mask. "
    ::= {slmLogControlEntry 6}

slmLogControlTypes OBJECT-TYPE
    SYNTAX EventType
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " This variable defines the type of events being logged. "
    ::= {slmLogControlEntry 7}

slmLogControlCause OBJECT-TYPE
    SYNTAX ProbableCause
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Any event with a different probable cause is ignored. "
    ::= {slmLogControlEntry 8}

slmLogControlSeverity OBJECT-TYPE
    SYNTAX PerceivedSeverity
    MAX-ACCESS read-create
    STATUS current
    DESCRIPTION
        " Any event with severity equal to or less the discriminated level
          is ignored. "
    ::= {slmLogControlEntry 9}

```

```

slmLogControlSpecificProblems OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        " Any event not generated by the identified object is ignored. "
    ::= {slmLogControlEntry 10}

slmLogControlToStateChange OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        " Any state changes to a different state than the one indicated are
          ignored. "
    ::= {slmLogControlEntry 11}

slmLogControlTrendIndication OBJECT-TYPE
    SYNTAX      TrendIndication
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        " Any events with a trend less or equal to specified value are
          ignored. "
    ::= {slmLogControlEntry 12}

slmLogControlChangedObjectId OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Any events not caused by a change of the value of this object
          are ignored. "
    ::= {slmLogControlEntry 13}

slmLogControlStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The control that allows creation/deletion of entries.
        Once made active an entry may not be modified except to. "
    ::= {slmLogControlEntry 14}

--
-- Alarm Log Group
--

slmAlarmLogTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SlmAlarmLogEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
        "A table of logged alarm events. "
    ::= {slmLogs 1}

slmAlarmLogEntry OBJECT-TYPE
    SYNTAX SlmAlarmLogEntry
    MAX-ACCESS not-accessible
    STATUS      current
    DESCRIPTION
        "A single alarm log. "
    INDEX {slmAlarmLogTime}
    ::= {slmAlarmLogTable 1}

SlmAlarmLogEntry ::= SEQUENCE {
    slmAlarmLogName      EntryName,
    slmAlarmLogTime      TimeTicks,
    slmAlarmLogText      DisplayString,
    slmAlarmLogType      EventType,
    slmAlarmLogCause      ProbableCause,
    slmAlarmLogSeverity   PerceivedSeverity,
    slmAlarmLogSpecificProblems OBJECT IDENTIFIER,
    slmAlarmLogTrendIndication TrendIndication,
    slmAlarmLogChangedObjectId OBJECT IDENTIFIER
}

```

```

}

slmAlarmLogName OBJECT-TYPE
    SYNTAX EntryName
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmAlarmLogEntry 1}

slmAlarmLogTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmAlarmLogEntry 2}

slmAlarmLogText OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "A description of the event's function and use. "
    DEFVAL {'H'}
    ::= {slmAlarmLogEntry 3}

slmAlarmLogType OBJECT-TYPE
    SYNTAX EventType
        MAX-ACCESS read-only
        STATUS current
    DESCRIPTION
        " Indicates the type of the event. "
    ::= {slmAlarmLogEntry 4}

slmAlarmLogCause OBJECT-TYPE
    SYNTAX ProbableCause
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "This variable defines further probable cause for
        the event. "
    ::= {slmAlarmLogEntry 5}

slmAlarmLogSeverity OBJECT-TYPE
    SYNTAX PerceivedSeverity
        MAX-ACCESS read-only
        STATUS current
    DESCRIPTION
        "This parameter defines the Perceived severity of the
        event. "
    ::= {slmAlarmLogEntry 6}

slmAlarmLogSpecificProblems OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        " This variable identifies the object responsible for the event. "
    ::= {slmAlarmLogEntry 7}

slmAlarmLogTrendIndication OBJECT-TYPE
    SYNTAX TrendIndication
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Indicates the trend of the event. "
    ::= {slmAlarmLogEntry 8}

slmAlarmLogChangedObjectId OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        " The object identifier of the object which changed value. "
    ::= {slmAlarmLogEntry 9}

```

```

--
-- State Logs Group
--

slmStateChangeLogTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SlmStateChangeLogEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A table of logged stateChange events. "
    ::= {slmLogs 2}

slmStateChangeLogEntry OBJECT-TYPE
    SYNTAX SlmStateChangeLogEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A single stateChange log. "
    INDEX {slmStateChangeLogTime}
    ::= {slmStateChangeLogTable 1}

SlmStateChangeLogEntry ::= SEQUENCE {
    slmStateChangeLogName      EntryName,
    slmStateChangeLogTime      TimeTicks,
    slmStateChangeLogText      DisplayString,
    slmStateChangeLogToStateChange Unsigned32,
    slmStateChangeLogChangedObjectId OBJECT IDENTIFIER
}

slmStateChangeLogName OBJECT-TYPE
    SYNTAX EntryName
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmStateChangeLogEntry 1}

slmStateChangeLogTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmStateChangeLogEntry 2}

slmStateChangeLogText OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "A description of the event's function and use. "
    DEFVAL {'H'}
    ::= {slmStateChangeLogEntry 3}

slmStateChangeLogToStateChange OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "If ChangedObjectId is a state/status/variable,
        this variable identifies the state that caused the
        event to be generated. "
    ::= {slmStateChangeLogEntry 4}

slmStateChangeLogChangedObjectId OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The object identifier of the MIB object that caused the event. "
    ::= {slmStateChangeLogEntry 5}

--
-- Object Value Change Logs Group
--

```

```

slmValueChangeLogTable OBJECT-TYPE
    SYNTAX SEQUENCE OF SlmValueChangeLogEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A table of logged valuechange events. "
    ::= {slmLogs 3}

slmValueChangeLogEntry OBJECT-TYPE
    SYNTAX SlmValueChangeLogEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        "A single valuechange log. "
    INDEX {slmValueChangeLogTime}
    ::= {slmValueChangeLogTable 1}

SlmValueChangeLogEntry ::= SEQUENCE {
    slmValueChangeLogName      EntryName,
    slmValueChangeLogTime      TimeTicks,
    slmValueChangeLogText      DisplayString,
    slmValueChangeLogChangedObjectId OBJECT IDENTIFIER
}

slmValueChangeLogName OBJECT-TYPE
    SYNTAX EntryName
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmValueChangeLogEntry 1}

slmValueChangeLogTime OBJECT-TYPE
    SYNTAX TimeTicks
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The name of the logged event. "
    ::= {slmValueChangeLogEntry 2}

slmValueChangeLogText OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "A description of the event's function and use. "
    DEFVAL {'H'}
    ::= {slmValueChangeLogEntry 3}

slmValueChangeLogChangedObjectId OBJECT-TYPE
    SYNTAX OBJECT IDENTIFIER
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "The object identifier of the MIB object that caused the event. "
    ::= {slmValueChangeLogEntry 4}

--
-- Conformance Information
--

slmCompliances      OBJECT IDENTIFIER ::= {slmMIBConformance 1}
slmGroups           OBJECT IDENTIFIER ::= {slmMIBConformance 2}

slmECMGCompliance  MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        " The compliance statement for SNMP Entities which host or
        represent ECMGs. A hosting entity must also support either
        threshold, state change, or value change logs. "
    MODULE -- this module
    GROUP slmThresholdLogControlGroup
    DESCRIPTION
        " This group is required if threshold logs are supported. "
    GROUP slmThresholdLogControlOptGroup
    DESCRIPTION

```

```

    " This group is optional if threshold logs are supported. "
GROUP slmStateChangeLogControlGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmStateChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional if stateChange logs are supported. "
GROUP slmValueChangeLogControlGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
GROUP slmValueChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional valueChange logs are supported. "
GROUP slmAlarmLogGroup
DESCRIPTION
    " This group is required if alarm logs are supported. "
GROUP slmAlarmLogOptGroup
DESCRIPTION
    " This group is optional if alarm logs are supported. "
GROUP slmStateChangeLogGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmValueChangeLogGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
::= {slmCompliances 1}

slmEmOrPdCompliance MODULE-COMPLIANCE
STATUS current
DESCRIPTION
    " The compliance statement for SNMP Entities which host or
    represent EMMG/PDGs. A hosting entity must also support either
    threshold, state change, or value change logs. "
MODULE -- this module
GROUP slmThresholdLogControlGroup
DESCRIPTION
    " This group is required if threshold logs are supported. "
GROUP slmThresholdLogControlOptGroup
DESCRIPTION
    " This group is optional if threshold logs are supported. "
GROUP slmStateChangeLogControlGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmStateChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional if stateChange logs are supported. "
GROUP slmValueChangeLogControlGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
GROUP slmValueChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional valueChange logs are supported. "
GROUP slmAlarmLogGroup
DESCRIPTION
    " This group is required if alarm logs are supported. "
GROUP slmAlarmLogOptGroup
DESCRIPTION
    " This group is optional if alarm logs are supported. "
GROUP slmStateChangeLogGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmValueChangeLogGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
::= {slmCompliances 2}

slmCpsigCompliance MODULE-COMPLIANCE
STATUS current
DESCRIPTION
    " The compliance statement for SNMP Entities which host or
    represent CPSIGs. A hosting entity must also support either
    threshold, state change, or value change logs. "
MODULE -- this module
GROUP slmThresholdLogControlGroup
DESCRIPTION
    " This group is required if threshold logs are supported. "
GROUP slmThresholdLogControlOptGroup
DESCRIPTION

```



```

    " This group is optional if threshold logs are supported. "
GROUP slmStateChangeLogControlGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmStateChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional if stateChange logs are supported. "
GROUP slmValueChangeLogControlGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
GROUP slmValueChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional valueChange logs are supported. "
GROUP slmAlarmLogGroup
DESCRIPTION
    " This group is required if alarm logs are supported. "
GROUP slmAlarmLogOptGroup
DESCRIPTION
    " This group is optional if alarm logs are supported. "
GROUP slmStateChangeLogGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmValueChangeLogGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
::= {slmCompliances 3}

slmCsigCompliance    MODULE-COMPLIANCE
STATUS current
DESCRIPTION
    " The compliance statement for SNMP Entities which host or
represent CSIGs. A hosting entity must also support either
threshold, state change, or value change logs. "
MODULE -- this module
GROUP slmThresholdLogControlGroup
DESCRIPTION
    " This group is required if threshold logs are supported. "
GROUP slmThresholdLogControlOptGroup
DESCRIPTION
    " This group is optional if threshold logs are supported. "
GROUP slmStateChangeLogControlGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmStateChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional if stateChange logs are supported. "
GROUP slmValueChangeLogControlGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
GROUP slmValueChangeLogControlOptGroup
DESCRIPTION
    " This Group is optional valueChange logs are supported. "
GROUP slmAlarmLogGroup
DESCRIPTION
    " This group is required if alarm logs are supported. "
GROUP slmAlarmLogOptGroup
DESCRIPTION
    " This group is optional if alarm logs are supported. "
GROUP slmStateChangeLogGroup
DESCRIPTION
    " This Group is required if stateChange logs are supported. "
GROUP slmValueChangeLogGroup
DESCRIPTION
    " This Group is required valueChange logs are supported. "
::= {slmCompliances 4}

slmAlarmLogGroup    OBJECT-GROUP
OBJECTS {
    slmAlarmLogName,
    slmAlarmLogTime,
    slmAlarmLogText,
    slmAlarmLogType,
    slmAlarmLogChangedObjectId
}
STATUS current
DESCRIPTION
    " A collection of objects specifying alarm logs. "
::= {slmGroups 1}

```

```

slmAlarmLogOptGroup OBJECT-GROUP
  OBJECTS {
    slmAlarmLogCause,
    slmAlarmLogSeverity,
    slmAlarmLogTrendIndication,
    slmAlarmLogSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying alarm logs. "
    ::= {slmGroups 2}

slmStateChangeLogGroup OBJECT-GROUP
  OBJECTS {
    slmStateChangeLogName,
    slmStateChangeLogTime,
    slmStateChangeLogText,
    slmStateChangeLogChangedObjectId,
    slmStateChangeLogToStateChange
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying state change logs. "
    ::= {slmGroups 3}

slmValueChangeLogGroup OBJECT-GROUP
  OBJECTS {
    slmValueChangeLogName,
    slmValueChangeLogTime,
    slmValueChangeLogText,
    slmValueChangeLogChangedObjectId
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying value change logs. "
    ::= {slmGroups 4}

slmThresholdLogControlGroup OBJECT-GROUP
  OBJECTS {
    slmLogControlName,
    slmLogControlTypes,
    slmLogControlCause ,
    slmLogControlSeverity,
    slmLogControlTrendIndication ,
    slmLogControlChangedObjectId,
    slmLogControlStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying threshold LOGCONTROLS. "
    ::= {slmGroups 5}

slmThresholdLogControlOptGroup OBJECT-GROUP
  OBJECTS {
    slmLogControlStartTime,
    slmLogControlStopTime,
    slmLogControlDailyStartTime,
    slmLogControlDailyStopTime,
    slmLogControlWeeklyMask,
    slmLogControlSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying threshold LOGCONTROLS. "
    ::= {slmGroups 6}

```

```

slmStateChangeLogControlGroup    OBJECT-GROUP
  OBJECTS {
    slmLogControlName,
    slmLogControlTypes,
    slmLogControlToStateChange,
    slmLogControlChangedObjectId,
    slmLogControlStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying state change LOGCONTROLS. "
    ::= {slmGroups 7}

slmStateChangeLogControlOptGroup  OBJECT-GROUP
  OBJECTS {
    slmLogControlStartTime,
    slmLogControlStopTime,
    slmLogControlDailyStartTime,
    slmLogControlDailyStopTime,
    slmLogControlWeeklyMask,
    slmLogControlSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying state change LOGCONTROLS. "
    ::= {slmGroups 8}

slmValueChangeLogControlGroup    OBJECT-GROUP
  OBJECTS {
    slmLogControlName,
    slmLogControlTypes,
    slmLogControlChangedObjectId,
    slmLogControlStatus
  }
  STATUS current
  DESCRIPTION
    " A collection of objects specifying value change LOGCONTROLS. "
    ::= {slmGroups 9}

slmValueChangeLogControlOptGroup  OBJECT-GROUP
  OBJECTS {
    slmLogControlStartTime,
    slmLogControlStopTime,
    slmLogControlDailyStartTime,
    slmLogControlDailyStopTime,
    slmLogControlWeeklyMask,
    slmLogControlSpecificProblems
  }
  STATUS current
  DESCRIPTION
    " A collection of optional objects specifying value change LOGCONTROLS. "
    ::= {slmGroups 10}

END

```

Annex L (normative): SIMCOMP⇌MUXCONFIG XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!--SIMCOMP-MUXCONFIG interface schema version 4-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="channel_close">
    <xs:complexType>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="channel_error">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="response"/>
      </xs:sequence>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="channel_setup">
    <xs:complexType>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
      <xs:attribute name="protocol_version" type="xs:unsignedByte" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="channel_status">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="response"/>
        <xs:element ref="protocols_supported" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="channel_test">
    <xs:complexType>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="channel_setup" minOccurs="0"/>
        <xs:element ref="channel_status" minOccurs="0"/>
        <xs:element ref="channel_test" minOccurs="0"/>
        <xs:element ref="channel_close" minOccurs="0"/>
        <xs:element ref="channel_error" minOccurs="0"/>
        <xs:element ref="TS_resource_request" minOccurs="0"/>
        <xs:element ref="TS_resource_update" minOccurs="0"/>
        <xs:element ref="TS_resource_discover" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="response">
    <xs:complexType>
      <xs:attribute name="response_code" type="xs:unsignedShort" use="required"/>
      <xs:attribute name="description" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="TS_resource_request">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="resource_identifier" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="TS_resource_discover">
    <xs:complexType>
      <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="TS_resource_update">
    <xs:complexType>
```

```

    <xs:sequence>
      <xs:element ref="response" minOccurs="0"/>
      <xs:element ref="resource_mapping" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="channel_id" type="xs:unsignedShort" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="resource_identifier">
  <xs:complexType>
    <xs:attribute name="original_network_id" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="transport_stream_id" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="mux_protocol_type" type="xs:unsignedByte" use="required"/>
    <xs:attribute name="client_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="resource_mapping">
  <xs:complexType>
    <xs:attribute name="original_network_id" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="transport_stream_id" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="mux_protocol_type" type="xs:unsignedByte" use="required"/>
    <xs:attribute name="IP_address" type="xs:string" use="required"/>
    <xs:attribute name="port_number" type="xs:unsignedShort" use="required"/>
    <xs:attribute name="is_allocated" type="xs:boolean" use="required"/>
    <xs:attribute name="client_id" type="xs:unsignedInt" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="protocols_supported">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="protocol_type" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="protocol_type" type="xs:unsignedByte"/>
</xs:schema>

```

Annex M (normative):

ACG⇔EIS XML Schema Definition

```

<?xml version="1.0" encoding="UTF-8"?>
<!--ACG-EIS interface schema version 4-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:simpleType name="ACG_access_criteria_type">
    <xs:restriction base="xs:hexBinary"/>
  </xs:simpleType>
  <xs:simpleType name="AC_ID_type">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
  <xs:simpleType name="AC_parameter_tag_type">
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="channel_ID_type">
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="AC_expiration_time_type">
    <xs:restriction base="xs:dateTime"/>
  </xs:simpleType>
  <xs:simpleType name="event_duration_type">
    <xs:restriction base="xs:duration">
      <xs:pattern value="PT[0-9]{2}H[0-5][0-9]M[0-5][0-9]S"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="error_information_type">
    <xs:restriction base="xs:string">
      <xs:maxLength value="65535"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="error_status_type">
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="max_comp_time_type">
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="MSG_ID_type">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
  <xs:simpleType name="private_CA_element_type">
    <xs:restriction base="xs:hexBinary"/>
  </xs:simpleType>
  <xs:simpleType name="protocol_version_type">
    <xs:restriction base="xs:unsignedByte"/>
  </xs:simpleType>
  <xs:simpleType name="super_CAS_ID_type">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
  <xs:complexType name="commercial_event_info_type" abstract="true">
    <xs:annotation>
      <xs:documentation>This abstract type shall be redefined in the context of each
implementation</xs:documentation>
    </xs:annotation>
  </xs:complexType>
  <xs:complexType name="access_criteria_structure_type">
    <xs:sequence>
      <xs:element name="ACG_access_criteria" type="ACG_access_criteria_type"/>
      <xs:element name="AC_parameter" type="AC_parameter_type" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="private_CA_element" type="private_CA_element_type" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="AC_ID" type="AC_ID_type" use="optional"/>
    <xs:attribute name="AC_expiration_time" type="AC_expiration_time_type" use="optional"/>
  </xs:complexType>
  <xs:complexType name="AC_parameter_type">
    <xs:attribute name="AC_parameter_tag" type="AC_parameter_tag_type" use="required"/>
  </xs:complexType>
  <xs:complexType name="channel_setup_message_type">
    <xs:attribute name="MSG_ID" type="MSG_ID_type" use="required"/>
    <xs:attribute name="super_CAS_ID" type="super_CAS_ID_type" use="required"/>
    <xs:attribute name="parameter_flag" type="xs:boolean" use="required"/>
    <xs:attribute name="interrupt_flag" type="xs:boolean" use="required"/>
  </xs:complexType>

```

```

<xs:complexType name="channel_test_message_type">
  <xs:attribute name="msg_ID" type="xs:unsignedShort" use="required"/>
</xs:complexType>
<xs:complexType name="channel_close_message_type">
  <xs:attribute name="msg_ID" type="xs:unsignedShort" use="required"/>
</xs:complexType>
<xs:complexType name="channel_status_message_type">
  <xs:attribute name="MSG_ID" type="MSG_ID_type" use="required"/>
  <xs:attribute name="max_comp_time" type="max_comp_time_type" use="required"/>
  <xs:attribute name="private_CA_element_flag" type="xs:boolean" use="required"/>
  <xs:attribute name="interrupt_flag" type="xs:boolean" use="required"/>
  <xs:attribute name="parameters_flag" type="xs:boolean" use="required"/>
</xs:complexType>
<xs:complexType name="channel_error_message_type">
  <xs:sequence>
    <xs:element name="error_status" type="error_status_type" maxOccurs="unbounded"/>
    <xs:element name="error_information" type="error_information_type" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="msg_ID" type="xs:unsignedShort" use="required"/>
</xs:complexType>
<xs:complexType name="AC_request_message_type">
  <xs:sequence>
    <xs:element name="commercial_event_info" type="commercial_event_info_type"/>
  </xs:sequence>
  <xs:attribute name="msg_ID"/>
</xs:complexType>
<xs:complexType name="AC_response_message_type">
  <xs:sequence>
    <xs:element name="access_criteria_structure" type="access_criteria_structure_type"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="MSG_ID" type="MSG_ID_type" use="required"/>
</xs:complexType>
<xs:complexType name="AC_interrupt_message_type">
  <xs:sequence>
    <xs:element name="ACG_access_criteria" type="ACG_access_criteria_type"/>
    <xs:element name="AC_parameter" type="AC_parameter_type" minOccurs="0"/>
    <xs:element name="private_CA_element" type="private_CA_element_type"
minOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="MSG_ID" type="MSG_ID_type" use="required"/>
  <xs:attribute name="AC_ID" type="AC_ID_type" use="required"/>
  <xs:attribute name="AC_expiration_time" type="AC_expiration_time_type" use="optional"/>
</xs:complexType>
<xs:complexType name="AC_error_message_type">
  <xs:sequence>
    <xs:element name="error_status" type="error_status_type" maxOccurs="unbounded"/>
    <xs:element name="error_information" type="error_information_type" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="msg_ID" type="xs:unsignedShort" use="required"/>
</xs:complexType>
<xs:complexType name="message_header_type">
  <xs:attribute name="protocol_version" type="protocol_version_type" use="required"/>
  <xs:attribute name="ACG_channel_ID" type="channel_ID_type" use="required"/>
</xs:complexType>
<xs:complexType name="EIS_message_type">
  <xs:choice>
    <xs:element name="channel_setup" type="channel_setup_message_type"/>
    <xs:element name="channel_test" type="channel_test_message_type"/>
    <xs:element name="channel_status" type="channel_status_message_type"/>
    <xs:element name="channel_close" type="channel_close_message_type"/>
    <xs:element name="channel_error" type="channel_error_message_type"
maxOccurs="unbounded"/>
  </xs:choice>
  <xs:choice>
    <xs:element name="AC_request" type="AC_request_message_type" maxOccurs="unbounded"/>
  </xs:choice>
</xs:complexType>
<xs:complexType name="ACG_message_type">
  <xs:choice>
    <xs:element name="channel_test" type="channel_test_message_type"/>
    <xs:element name="channel_status" type="channel_status_message_type"/>
    <xs:element name="channel_error" type="channel_error_message_type"
maxOccurs="unbounded"/>
  </xs:choice>
  <xs:choice>
    <xs:element name="AC_interrupt" type="AC_interrupt_message_type"
maxOccurs="unbounded"/>
  </xs:choice>
  <xs:sequence>

```

```

        <xs:element name="AC_response" type="AC_response_message_type" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="AC_error" type="AC_error_message_type" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:choice>
</xs:choice>
</xs:complexType>
<xs:element name="EIS_ACG_message">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="msg_header" type="message_header_type"/>
            <xs:choice>
                <xs:element name="EIS_message" type="EIS_message_type"/>
                <xs:element name="ACG_message" type="ACG_message_type"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```


Annex N (normative):

ECMG ↔ SCS and EMMG ↔ MUX interfaces adaptations for use with IP Datacasting over DVB-H

N.1 Introduction

In the context of IP Datacasting over DVB-H, different scrambling algorithms based on AES-128 are used instead of the DVB-CSA algorithm:

- ISMACryp: specifies an end-to-end encryption system for ISMA 1.0 and 2.0 streams. ISMACryp makes use of a 128 bit AES key to perform encryption of the content. This key **SHOULD** change rapidly over time (typically every few seconds) and is identified by the ISMACryp **Key Indicator** (KI). The details of the ISMACryp scrambling algorithm are defined in the ISMA Encryption and Authentication specification [27] and the specific use for IP Datacasting over DVB-H is described in the Service Purchase and Protection specification [26].
- Internet Protocol Security (IPSec): it provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. IPsec ESP makes use of a 128 bit AES key to perform encryption of the content. This key **SHOULD** change rapidly over time (typically every few seconds) and is identified by the **Security Parameter Index** (SPI) carried in the ESP header. The details of the IPsec scrambling algorithm are defined in [28] and the specific use for IP Datacasting over DVB-H is described in the Service Purchase and Protection specification [26].
- Secure Real-Time Protocol (SRTP): it is a security profile for RTP that adds confidentiality, message authentication, and replay protection to the RTP protocol. SRTP makes use of a 128 bit AES key, derived from a 128 AES Master Key to perform encryption of the content. This key **SHOULD** change rapidly over time (typically every few seconds) and is identified by the **Master Key Indicator** (MKI) carried in the SRTP header. The details of the SRTP scrambling algorithm are defined in RFC 3711 [29] and the specific use for IP Datacasting over DVB-H is described in the Service Purchase and Protection specification [26].

The CW used in the DVB-CSA context is replaced by the functionally equivalent Traffic Encryption Key (TEK), which is used in the IP Datacasting over DVB-H environment. In order to accommodate the 128-bit AES key, the TEK is 16 bytes long, rather than the 8 bytes of the CW.

N.2 CP_CW_combination parameter in ECMG ↔ SCS

In order to match the different DVB-H scrambling requirements, the definition of the CP_CW_combination parameter in table 5 and clause 5.3 is changed to the following.

Table N.1: CP_CW_Combination parameter_type values for use with IPDC

Parameter_type Value	Parameter Type	Type/Units	Length (bytes)
0x0014	CP_CW_Combination	---	variable
	CP	uimbsf	2
	KSI	uimbsf	Variable
	TKM	uimbsf	16 or 32

CP_CW_combination: this parameter is the concatenation of the Crypto period number, the Key System Information (KSI) and the Traffic Key Material (TKM) from which the TEK and the optional Traffic Authentication Key (TAK) are derived. The meaning and length of the KSI varies according to the scrambling method used:

- In the case of ISMACryp, the KSI is the *Key Indicator*, and $1 \leq \text{KSI length} \leq 255$ bytes.
- In the case of IPsec, the KSI is the *Security Parameter Index*, and KSI length = 4 bytes.

- In the case of SRTP, the KSI is the *Master Key Indicator*, and $1 \leq \text{KSI length} \leq 255$ bytes.

The meaning and length of the TKM varies according to the scrambling method used:

- In the case of ISMACryp, the TKM is identical to the TEK, and the TKM length = 16 bytes.
- In the case of IPsec without traffic authentication, the TKM is identical to the TEK, and the TKM length = 16 bytes.
- In the case of IPsec with traffic authentication, the TKM is a concatenation of the TEK (16 bytes) and the *Traffic Authentication Seed* (16 bytes), and the TKM length = 32 bytes. The IPsec *Traffic Authentication Key* (20 bytes) is derived from the IPsec *Traffic Authentication Seed* as described in clause N.5.
- In the case of SRTP, the TKM is the *Master Key*, and TKM length = 16 bytes. The mechanism by which the TEK and TAK are derived from the *Master Key* is as defined by SRTP [29].

The length of the variable sized KSI parameters can be calculated by subtracting the length of the CP parameter (2 bytes) and the length of the TKM parameter (16 or 32 bytes) from the overall length of the CP_CW_Combination parameter.

N.3 Section_TSpkt_flag parameter in ECMG⇔SCS

In the context of IP Datacasting over DVB-H, ECMs/KSMs are arbitrary-length datagrams. The ECMG SHALL therefore set the *section_TSpkt_flag* parameter in the *channel_status* message to 0x02, and provide the ECM/KSM in the *ECM_response* message as an arbitrary-length datagram suitable for direct encapsulation as the payload of a UDP packet inserted in the broadcast bitstream.

Refer to clause 5.3 for further information on the use of the *section_TSpkt_flag* parameter in the ECMG⇔SCS interface.

N.4 Section_TSpkt_flag parameter in EMMG⇔MUX

In the context of IP Datacasting over DVB-H, EMMs/KMMs are delivered to the IP Encapsulator using one of two mechanism: the EMMs/KMMs are either sent to the IP Encapsulator as a unicast stream, or the EMMs/KMMs are delivered to the IP Encapsulator using the EMMG⇔MUX protocol as defined in this document. In the latter case, the *section_TSpkt_flag* parameter in the *channel_setup* and *channel_status* messages SHALL be set to 0x02, and EMM/KMM in the *data_provision* message SHALL be an arbitrary-length datagram suitable for direct encapsulation as the payload of a UDP packet inserted in the broadcast bitstream.

Refer to clause 6.2.3 for further information on the use of the *section_TSpkt_flag* parameter in the EMMG⇔MUX interface.

N.5 IPsec Traffic Authentication Key Derivation

If the IPsec scrambling method is used in combination with the optional traffic authentication feature, then the applicable IPsec *Traffic Authentication Key* shall be derived from the IPsec *Traffic Authentication Seed* according to the following mechanism:

- A pseudo-random function based on AES and with an output block size of 128 bits is used in the key generation process. This function, denoted *AES-XCBC-MAC-PRF*{key}(text), is defined in RFC 3566 [30] and RFC 4434 [31].
- The *AES-XCBC-MAC-PRF* function is used in the following manner to produce the required pseudo-random key material:
 - $T1 = \text{AES-XCBC-MAC-PRF}\{\text{Traffic Authentication Seed}\}(\text{CONSTANT_TAK} \parallel 0x01)$
 - $T2 = \text{AES-XCBC-MAC-PRF}\{\text{Traffic Authentication Seed}\}(T1 \parallel \text{CONSTANT_TAK} \parallel 0x02)$

Where $CONSTANT_TAK = 0x04040404040404040404040404040404$ (120 bits), and "||" denotes the concatenation operation.

- The 160-bit *Traffic Authentication Key* is extracted from the generated pseudo-random key material in the following manner:
 - $Traffic\ Authentication\ Key = MSB_{160}(T1||T2)$

Where $MSB_{160}()$ is a function that returns the 160 most significant bits from the supplied input, and "||" denotes the concatenation operation.

Annex O (informative): Bibliography

- FIPS 81 (1980): "DES Modes of Operation".
- ANSI X3.92 (1981): "Data Encryption Algorithm".
- ANSI X3.106 (1983): "Data Encryption Algorithm, Modes of Operation".
- Schneier, Bruce: "Applied Cryptography, Second Edition".
- Menezes, van Oorschot, Vanstone: "Handbook of Applied Cryptography".
- IETF RFC 1157 (1990): "Simple Network Management Protocol (SNMP)".
- ITU-T Recommendation X.209 (1988): "Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)".
- ISO/IEC 8825 (1990): "Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)".

History

Document history		
V1.1.1	June 2000	Publication
V1.2.1	January 2002	Publication
V1.3.1	January 2003	Publication
V1.4.1	September 2004	Publication
V1.5.1	October 2008	Publication