International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.248.1

(09/2005)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Communication procedures

Gateway control protocol: Version 3

ITU-T Recommendation H.248.1

ITU-T H-SERIES RECOMMENDATIONS

AUDIOVISUAL AND MULTIMEDIA SYSTEMS

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	**H.240–H.259**
Coding of moving video	H.260–H.279
Related systems aspects	H.280–H.299
Systems and terminal equipment for audiovisual services	H.300–H.349
Directory services architecture for audiovisual and multimedia services	H.350–H.359
Quality of service architecture for audiovisual and multimedia services	H.360–H.369
Supplementary services for multimedia	H.450–H.499
MOBILITY AND COLLABORATION PROCEDURES	
Overview of Mobility and Collaboration, definitions, protocols and procedures	H.500–H.509
Mobility for H-Series multimedia systems and services	H.510–H.519
Mobile multimedia collaboration applications and services	H.520–H.529
Security for mobile multimedia systems and services	H.530–H.539
Security for mobile multimedia collaboration applications and services	H.540–H.549
Mobility interworking procedures	H.550–H.559
Mobile multimedia collaboration inter-working procedures	H.560–H.569
BROADBAND AND TRIPLE-PLAY MULTIMEDIA SERVICES	
Broadband multimedia services over VDSL	H.610–H.619

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation H.248.1

Gateway control protocol: Version 3

Summary

To achieve greater scalability, this Recommendation decomposes the H.323 Gateway function defined in ITU-T Rec. H.246 into functional subcomponents and specifies the protocols these components use to communicate. This allows implementations of H.323 gateways to be highly scalable and encourages leverage of widely deployed Switched Circuit Network (SCN) capabilities such as SS7 switches. This also enables H.323 gateways to be composed of components from multiple vendors distributed across multiple physical platforms. The purpose of this Recommendation is to add capabilities currently defined for H.323 systems and is intended to provide new ways of performing operations already supported in H.323.

This Recommendation includes several enhancements to ITU-T Rec. H.248.1 Version 2:

- capability to define context properties via packages;
- an IEPS context property;
- a flag to indicate that the MG has OutOfService terminations to report at registration time;
- new message segmentation package and procedures for non-segmenting transports;
- refined package definition requirements and a new package template;
- refined profile definition requirements and a new profile template;
- addition of statistics on a stream level;
- addition of a signal request identifier to differentiate similar signals within a SignalList;
- addition of a base signal parameter to indicate in which direction to play the signal;
- addition of two new Topology types;
- addition of an intersignal delay timer for signals in a SignalList;
- addition of a new ContextIDList construct for command responses;
- addition of a TerminationIDList construct for commands and responses;
- refined ServiceChange procedures;
- addition of a capability for the MGC to regulate the rate at which it receives notifications;
- addition of the ability to add filter conditions to audit requests.

Source

ITU-T Recommendation H.248.1 was approved on 13 September 2005 by ITU-T Study Group 16 (2005-2008) under the ITU-T Recommendation A.8 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

ITU-T Recommendation H.248.1

Gateway control protocol: Version 3

1 Scope

This Recommendation defines the protocols used between elements of a physically decomposed multimedia gateway. There are no functional differences from a system view between a decomposed gateway, with distributed sub-components potentially on more than one physical device, and a monolithic gateway such as described in ITU-T Rec. H.246. This Recommendation does not define how gateways, multipoint control units or interactive voice response units (IVRs) work. Instead it creates a general framework that is suitable for these applications.

Packet network interfaces may include IP, ATM or possibly others. The interfaces will support a variety of Switched Circuit Network (SCN) signalling systems, including tone signalling, ISDN, ISUP, QSIG and GSM. National variants of these signalling systems will be supported where applicable.

Products claiming compliance with Version 1 of ITU-T Rec. H.248.1 shall comply with all of the mandatory requirements of ITU-T Rec. H.248.1 originally approved in 06/2000 and reissued in 03/2002.

Products claiming compliance with Version 2 of ITU-T Rec. H.248.1 shall comply with all of the mandatory requirements of ITU-T Rec. H.248.1 originally approved in 05/2002 and updated in 03/2004.

Products claiming compliance with this Recommendation shall comply with all of the mandatory requirements of ITU-T Rec. H.248.1 approved on 09/2005.

Products shall indicate the version of the protocol in use by using ServiceChangeVersion as '1' to refer to ITU-T Rec. H.248.1 (03/2002), '2' to refer to ITU-T Rec. H.248.1 Corrigendum 1 (03/2004) and '3' to refer to ITU-T Rec. H.248.1 (09/2005).

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

2.1 Normative references

– ITU-T Recommendation E.106 (2003), *International Emergency Preference Scheme (IEPS) for disaster relief operations*.

– ITU-T Recommendation H.225.0 (2003), *Call signalling protocols and media stream packetization for packet-based multimedia communication systems*.

– ITU-T Recommendation H.235.0 (2005), *H.323 Security: Framework for security in H-series (H.323 and other H.245-based) multimedia systems*.

– ITU-T Recommendation H.245 (2005), *Control protocol for multimedia communication*.

– ITU-T Recommendation H.246 (1998), *Interworking of H-series multimedia terminals with H-series multimedia terminals and voice/voiceband terminals on GSTN and ISDN*.

- ITU-T Recommendation H.248.4 (2000), *Gateway control protocol: Transport over Stream Control Transmission Protocol (SCTP)*, plus Corrigendum 1 (2004).
- ITU-T Recommendation H.248.5 (2000), *Gateway control protocol: Transport over ATM*.
- ITU-T Recommendation H.248.8 (2005), *Gateway control protocol: Error code and service change reason description*.
- ITU-T Recommendation H.248.14 (2002), *Gateway control protocol: Inactivity timer package*.
- ITU-T Recommendation H.323 (2003), *Packet-based multimedia communications systems*.
- ITU-T Recommendation I.363.1 (1996), *B-ISDN ATM Adaptation Layer specification: Type 1 AAL*.
- ITU-T Recommendation I.363.2 (2000), *B-ISDN ATM Adaptation Layer specification: Type 2 AAL*.
- ITU-T Recommendation I.363.5 (1996), *B-ISDN ATM Adaptation Layer specification: Type 5 AAL*.
- ITU-T Recommendation I.366.1 (1998), *Segmentation and Reassembly Service Specific Convergence Sublayer for the AAL type 2*.
- ITU-T Recommendation I.366.2 (2000), *AAL type 2 service specific convergence sublayer for narrow-band services*, plus Corrigendum 1 (2002).
- ITU-T Recommendation I.371 (2004), *Traffic control and congestion control in B-ISDN*.
- ITU-T Recommendation Q.763 (1999), *Signalling System No. 7 – ISDN user part formats and codes*, plus Amendment 3 (2004).
- ITU-T Recommendation Q.765.5 (2004), *Signalling System No. 7 – Application transport mechanism: Bearer Independent Call Control (BICC)*.
- ITU-T Recommendation Q.931 (1998), *ISDN user-network interface layer 3 specification for basic call control*, plus Amendment 1 (2002): *Extensions for the support of digital multiplexing equipment*.
- ITU-T Recommendation Q.2630.1 (1999), *AAL type 2 signalling protocol – Capability Set 1*.
- ITU-T Recommendation Q.2931 (1995), *Digital subscriber signalling system No. 2 – User-Network Interface (UNI) layer 3 specification for basic call/connection control*, plus Amendment 4 (1999).
- ITU-T Recommendation Q.2941.1 (1997), *Digital subscriber signalling system No. 2 – Generic identifier transport*.
- ITU-T Recommendation Q.2961.1 (1995), *Digital subscriber signalling system No. 2 – Additional traffic parameters: Additional signalling capabilities to support traffic parameters for the tagging option and the sustainable call rate parameter set*.
- ITU-T Recommendation Q.2961.2 (1997), *Digital subscriber signalling system No. 2 – Additional traffic parameters: Support of ATM transfer capability in the broadband bearer capability information element*, plus Corrigendum 1 (1999).
- ITU-T Recommendation Q.2965.1 (1999), *Digital subscriber signalling system No. 2 – Support of Quality of Service classes*, plus Amendment 1 (2000).

- ITU-T Recommendation Q.2965.2 (1999), *Digital subscriber signalling system No. 2 – Signalling of individual Quality of Service parameters*.
- ITU-T Recommendation V.76 (1996), *Generic multiplexer using V.42 LAPM-based procedures*, plus Corrigendum 1 (2005).
- ITU-T Recommendation X.213 (2001) | ISO/IEC 8348:2002, *Information technology – Open Systems Interconnection – Network service definition*.
- ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*, plus Amendment 2 (2004): *Alignment with changes made to Rec. X.660 | ISO/IEC 9834-1 for identifiers in object identifier value notation*.
- ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, *Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, plus Amendment 1 (2003): *Support for EXTENDED-XER*.
- ISO/IEC 10646 (2003), *Information technology – Universal Multiple-Octet Coded character Set (UCS)*.
- ATM Forum (1996), *ATM User-Network Interface (UNI) Signalling Specification – Version 4.0*.
- IETF RFC 1006 (1987), *ISO Transport Service on top of the TCP, Version 3*.
- IETF RFC 2234 (1997), *Augmented BNF for Syntax Specifications: ABNF*.
- IETF RFC 2327 (1998), *SDP: Session Description Protocol*.
- IETF RFC 2402 (1998), *IP Authentication Header*.
- IETF RFC 2406 (1998), *IP Encapsulating Security Payload (ESP)*.

2.2 Informative references

- ITU-T Recommendation E.180/Q.35 (1998), *Technical characteristics of tones for the telephone service*.
- ITU-T Recommendation G.711 (1988), *Pulse code modulation (PCM) of voice frequencies*.
- ITU-T Recommendation H.221 (2004), *Frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices*.
- ITU-T Recommendation H.223 (2001), *Multiplexing protocol for low bit rate multimedia communication*.
- ITU-T Recommendation H.226 (1998), *Channel aggregation protocol for multilink operation on circuit-switched networks*.
- ITU-T Recommendation Q.724 (1998), *Telephone user part signalling procedures*, plus Amendment 1 (1993).
- ITU-T Recommendation Q.764 (1999), *Signalling System No. 7 – ISDN user part signalling procedures*, plus Amendment 3 (2004).
- ITU-T Recommendation Q.1902.4 (2001), *Bearer Independent Call Control protocol – (Capability Set 2): Basic call procedures*, plus Amendment 2 (2004).
- IETF RFC 768 (1980), *User Datagram Protocol*.
- IETF RFC 791 (1981), *Internet protocol*.

– IETF RFC 793 (1981), *Transmission control protocol*.

– IETF RFC 1661 (1994), *The Point-to-Point Protocol (PPP)*.

– IETF RFC 2401 (1998), *Security Architecture for the Internet Protocol*.

– IETF RFC 2460 (1998), *Internet Protocol, Version 6 (IPv6) Specification*.

– IETF RFC 2805 (2000), *Media Gateway Control Protocol Architecture and Requirements*.

– IETF RFC 3261 (2002), *SIP: Session Initiation Protocol*.

– IETF RFC 3550 (2003), *RTP: A Transport Protocol for Real-Time Applications*.

– IETF RFC 3551 (2003), *RTP Profile for Audio and Video Conferences with Minimal Control*.

3 Definitions

This Recommendation defines the following terms:

3.1 access gateway: A type of gateway that provides a User-Network Interface (UNI) such as ISDN.

3.2 descriptor: A syntactic element of the protocol that groups related properties. For instance, the properties of a media flow on the MG can be set by the MGC by including the appropriate descriptor in a command.

3.3 Media Gateway (MG): The media gateway converts media provided in one type of network to the format required in another type of network. For example, a MG could terminate bearer channels from a switched circuit network (e.g., DS0s) and media streams from a packet network (e.g., RTP streams in an IP network). This gateway may be capable of processing audio, video and T.120 alone or in any combination, and will be capable of full duplex media translations. The MG may also play audio/video messages and perform other IVR functions, or may perform media conferencing.

3.4 Media Gateway Controller (MGC): Controls the parts of the call state that pertain to connection control for media channels in a MG.

3.5 Multipoint Control Unit (MCU): An entity that controls the setup and coordination of a multi-user conference that typically includes processing of audio, video and data.

3.6 residential gateway: A gateway that interworks an analogue line to a packet network. A residential gateway typically contains one or two analogue lines and is located at the customer premises.

3.7 SCN FAS signalling gateway: This function contains the SCN Signalling Interface that terminates SS7, ISDN or other signalling links where the call control channel and bearer channels are collocated in the same physical span.

3.8 SCN NFAS signalling gateway: This function contains the SCN Signalling Interface that terminates SS7 or other signalling links where the call control channels are separated from bearer channels.

3.9 stream: Bidirectional media or control flow received/sent by a media gateway as part of a call or conference.

3.10 trunk: A communication channel between two switching systems such as a DS0 on a T1 or E1 line.

3.11 trunking gateway: A gateway between SCN network and packet network that typically terminates a large number of digital circuits.

4 Abbreviations

This Recommendation uses the following abbreviations:

AAD	Average Acknowledgement Delay
AAL	ATM Adaptation Layer
ADEV	Average Deviation
ALF	Application Level Framing
ATM	Asynchronous Transfer Mode
C	Context
CAS	Channel Associated Signalling
DNS	Domain Name System
DTMF	Dual Tone Multi-Frequency
FAS	Facility Associated Signalling
GSM	Global System for Mobile communications
GW	Gateway
IANA	Internet Assigned Numbers Authority (superseded by ICANN)
ICANN	Internet Corporation for Assigned Names and Numbers
IEPS	International Emergency Preference Scheme
IP	Internet Protocol
IS	In-Service
ISUP	ISDN User Part
IVR	Interactive Voice Response
MG	Media Gateway
MGC	Media Gateway Controller
MWD	Maximum Waiting Delay
NFAS	Non-Facility Associated Signalling
OoS	Out-of-Service
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTP	Real-time Transport Protocol
SC	ServiceChange
SCN	Switched Circuit Network
SG	Signalling Gateway
SS7	Signalling System No. 7
T, Term	Termination

5 Conventions

In this Recommendation, "shall" refers to a mandatory requirement, while "should" refers to a suggested but optional feature or procedure. The term "may" refers to an optional course of action without expressing a preference.

6 Connection model

The connection model for the protocol describes the logical entities, or objects, within the Media Gateway that can be controlled by the Media Gateway Controller. The main abstractions used in the connection model are terminations and contexts.

A *termination* sources and/or sinks one or more streams. In a multimedia conference, a termination can be multimedia and sources or sinks multiple media streams. The media stream parameters, as well as bearer parameters are encapsulated within the termination.

A *context* is an association between a collection of terminations. There is a special type of context, the *NULL* Context, which contains all terminations that are not present in any other context and therefore not associated to any other termination. For instance, in a decomposed access gateway, all idle lines are represented by terminations in the NULL Context.

Following is a graphical depiction of these concepts. The diagram of Figure 1 gives several examples and is not meant to be an all-inclusive illustration. The asterisk box in each of the contexts represents the logical association of terminations implied by the context.

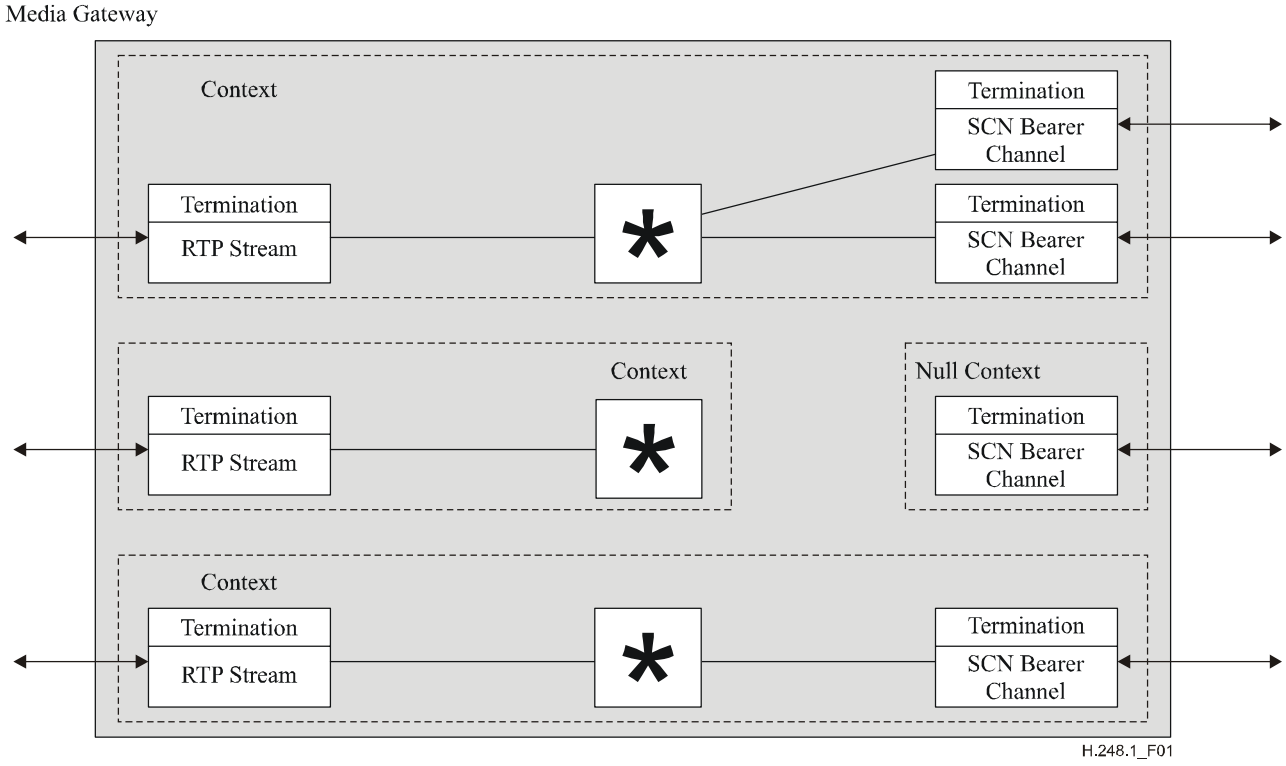

Figure 1/H.248.1 – Example of H.248.1 connection model

The example in Figure 2 shows an example of one way to accomplish a call-waiting scenario in a decomposed access gateway, illustrating the relocation of a termination between contexts. Terminations T1 and T2 belong to Context C1 in a two-way audio call. A second audio call is waiting for T1 from Termination T3. T3 is alone in Context C2. T1 accepts the call from T3, placing T2 on hold. This action results in T1 moving into Context C2, as shown in Figure 3.

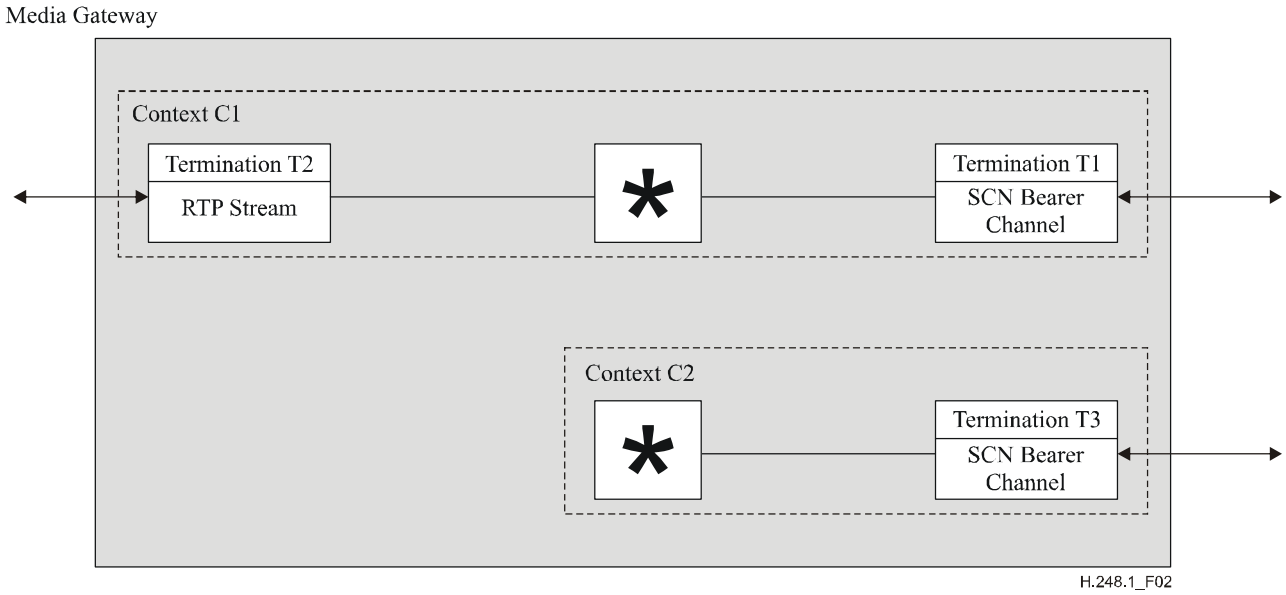

Figure 2/H.248.1 – Example call waiting scenario/alerting applied to T1

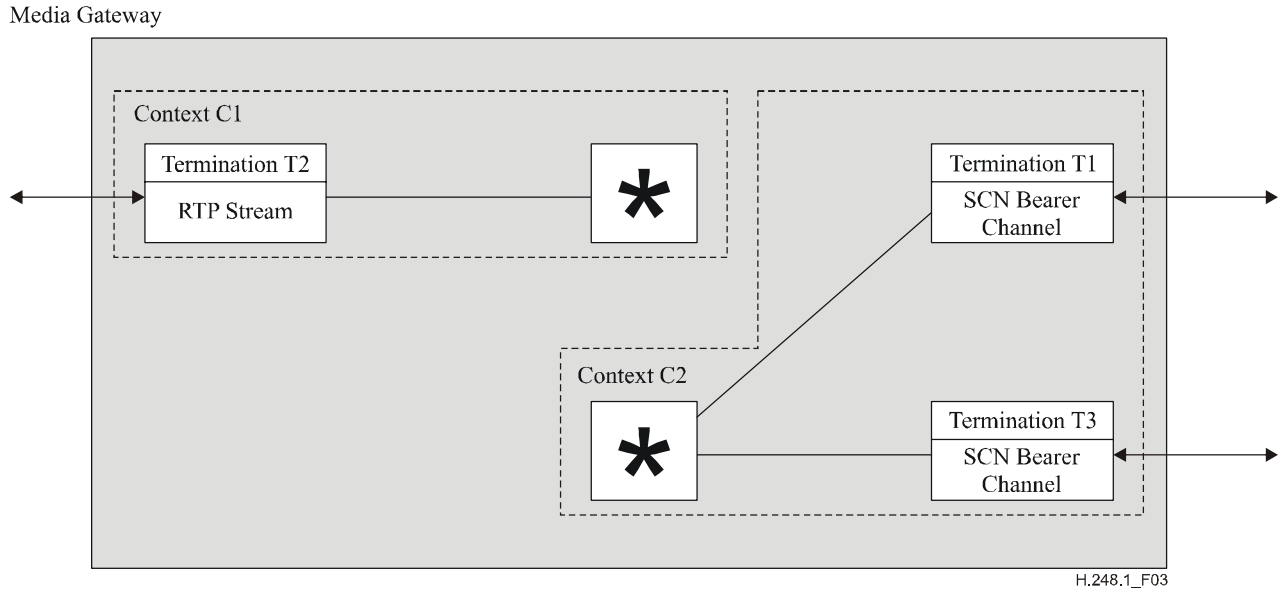

Figure 3/H.248.1 – Example call waiting scenario/answer by T1

6.1 Contexts

A context is an association between a number of terminations. The context describes the topology (who hears/sees whom) and the media mixing and/or switching parameters if more than two terminations are involved in the association.

There is a special context called the *NULL* Context. It contains terminations that are not present in any other context and therefore not associated to any other termination. Terminations in the NULL Context can have their parameters examined or modified, and may have events detected on them.

In general, an Add Command is used to add terminations to contexts. If the MGC does not specify an existing context to which the termination is to be added, the MG creates a new context. A termination may be removed from a context with a Subtract Command, and a termination may be moved from one context to another with a Move Command. A termination shall exist in only one context at a time.

The maximum number of terminations in a context is a MG property. Media gateways that offer only point-to-point connectivity might allow at most two terminations per context. Media gateways that support multipoint conferences might allow three or more terminations per context.

6.1.1 Context attributes and descriptors

The attributes of contexts are:

- ContextID.
- The Topology Descriptor (who hears/sees whom).

  The topology of a context describes the flow of media between the terminations within a context. In contrast, the Mode Property of a termination ("SendOnly"/"RecvOnly"/…) describes the flow of the media at the egress/ingress of the media gateway.
- The priority is used for a context in order to provide the MG with information about a certain precedence handling for a context. The MGC can also use the priority to control autonomously the traffic precedence in the MG in a smooth way in certain situations (e.g., restart), when a lot of contexts must be handled simultaneously. Priority 0 is the lowest priority and a priority of 15 is the highest priority.
- An indicator for an emergency call is also provided to allow a preference handling in the MG.
- An indicator for an IEPS call is provided to allow the features and techniques of E.106 to be achieved.
- A ContextAttribute Descriptor that enables extra context attributes to be defined by using the packages extension mechanism (see 7.1.19).

6.1.2 Creating, deleting and modifying contexts

The protocol can be used to (implicitly) create contexts and modify the parameter values of existing contexts. The protocol has commands to add terminations to contexts, subtract them from contexts, and to move terminations between contexts. Contexts are deleted implicitly when the last remaining termination is subtracted or moved out.

6.2 Terminations

A termination is a logical entity on a MG that sources and/or sinks media and/or control streams. A termination is described by a number of characterizing properties, which are grouped in a set of descriptors that are included in commands. Terminations have unique identities (TerminationIDs), assigned by the MG at the time of their creation.

Terminations representing physical entities have a semi-permanent existence. For example, a termination representing a TDM channel might exist for as long as it is provisioned in the gateway. Terminations representing ephemeral information flows, such as RTP flows, would usually exist only for the duration of their use.

Ephemeral terminations are created by means of an Add Command. They are destroyed by means of a Subtract Command. In contrast, when a physical termination is Added to or Subtracted from a context, it is taken from or to the NULL Context, respectively.

Terminations may have signals applied to them (see 7.1.11). Terminations may be programmed to detect events, the occurrence of which can trigger notification messages to the MGC, or action by the MG. Statistics may be accumulated on a termination. Statistics are reported to the MGC upon request (by means of the AuditValue Command, see 7.2.5) and when the termination ceases to exist or is returned to the NULL Context due to a Subtract Command.

Multimedia gateways may process multiplexed media streams. For example, ITU-T Rec. H.221 describes a frame structure for multiple media streams multiplexed on a number of digital 64 kbit/s

channels. Such a case is handled in the connection model in the following way. For every bearer channel that carries part of the multiplexed streams, there is a physical or ephemeral "bearer termination". The bearer terminations that source/sink the digital channels are connected to a separate termination called the "multiplexing termination". The multiplexing termination is an ephemeral termination representing a frame-oriented session. The Multiplex Descriptor for this termination describes the multiplex used (e.g., H.221 for an H.320 session) and indicates the order in which the contained digital channels are assembled into a frame.

Multiplexing terminations may be cascades (e.g., H.226 multiplex of digital channels feeding into a H.223 multiplex supporting an H.324 session).

The individual media streams carried in the session are described by Stream Descriptors on the multiplexing termination. These media streams can be associated with streams sourced/sunk by terminations in the context other than the bearer terminations supporting the multiplexing termination. Each bearer termination supports only a single data stream. These data streams do not appear explicitly as streams on the multiplexing termination and they are hidden from the rest of the context.

Figures 4, 5, and 6 illustrate typical applications of the multiplexing termination and Multiplex Descriptor.

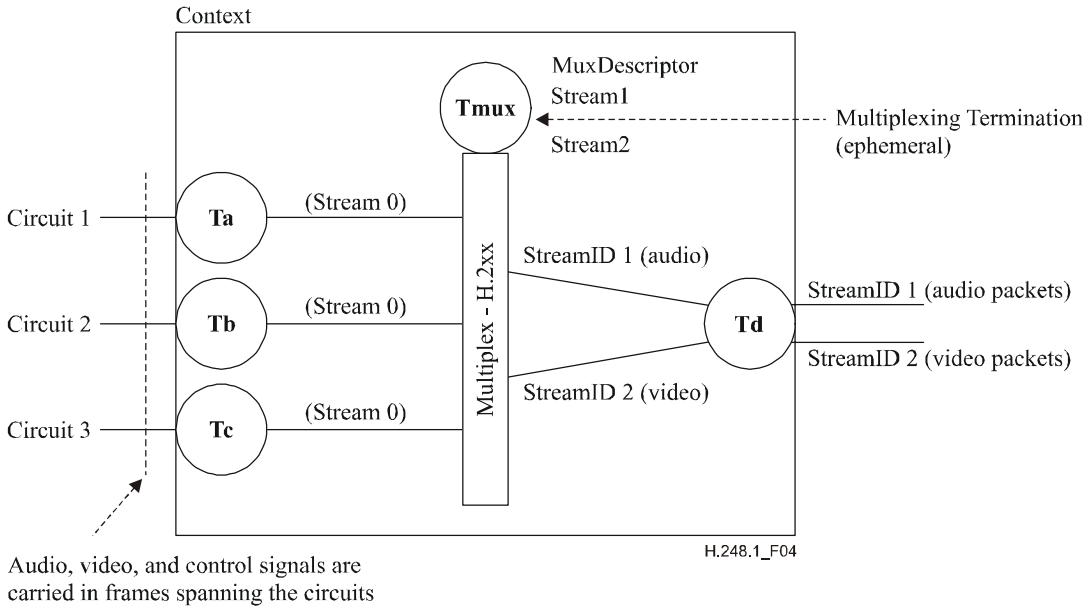

Figure 4/H.248.1 – Multiplexed termination scenario – Circuit-to-packet

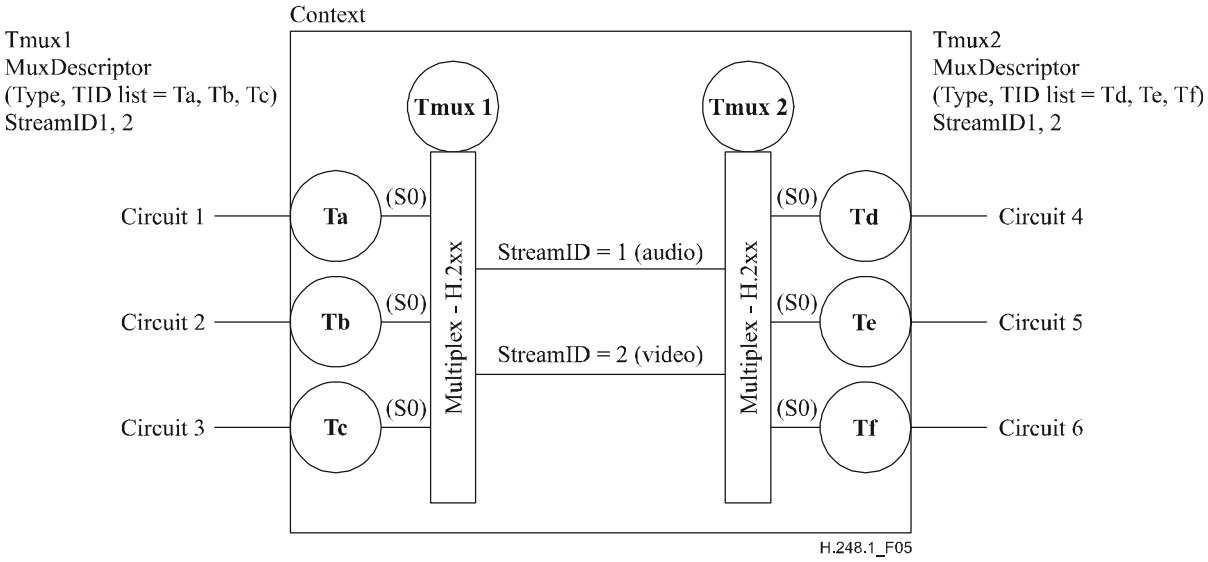

Figure 5/H.248.1 – Multiplexed termination scenario – Circuit-to-circuit

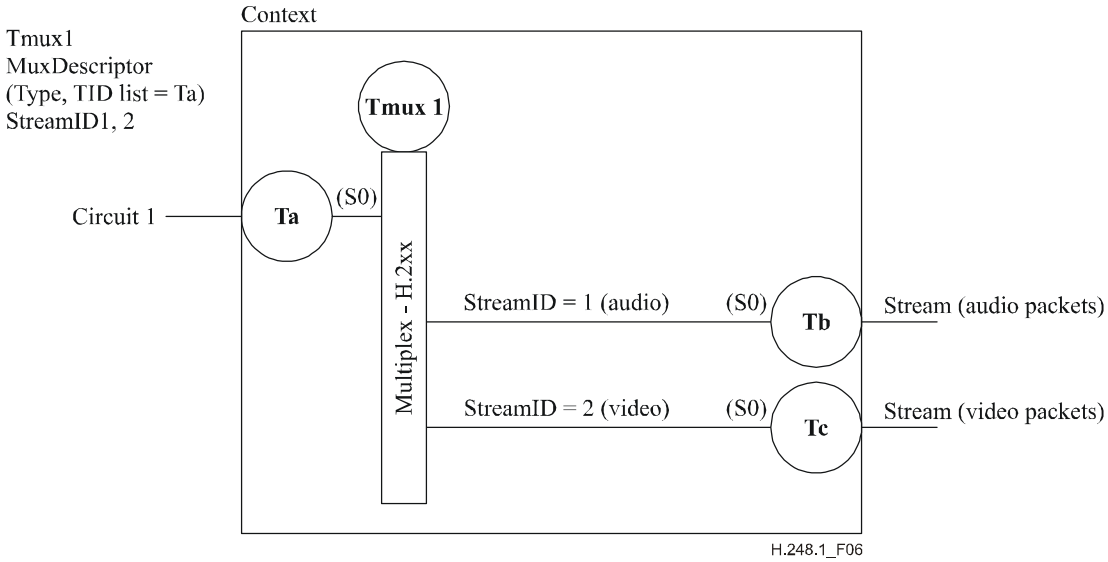

Figure 6/H.248.1 – Multiplexed termination scenario – Single-to-multiple terminations

Unlike the multiplexing terminations described in the previous paragraph the multiplexed bearer termination, which represent multiplexed bearers such as ATM AAL Type 2 bearers, carry no media streams. They are present strictly for the purpose of modeling the creation and destruction of the actual bearer. When a new multiplexed bearer is to be created, an ephemeral termination is created in a context established for this purpose. When the termination is subtracted, the multiplexed bearer is destroyed.

6.2.1 Termination dynamics

The protocol can be used to create new terminations and to modify property values of existing terminations. These modifications include the possibility of adding or removing events and/or signals. The termination properties, and events and signals are described in the ensuing subclauses. An MGC can only release/modify terminations and the resources that the termination represents which are in the NULL Context or which have been previously seized via e.g., the Add Command.

6.2.2 TerminationIDs

Terminations are referenced by a TerminationID, which is an arbitrary schema chosen by the MG.

TerminationIDs of physical terminations are provisioned in the Media Gateway. The TerminationIDs may be chosen to have structure. For instance, a TerminationID may consist of trunk group and a trunk within the group.

A wildcarding mechanism using two types of wildcards can be used with TerminationIDs. The two wildcards are ALL and CHOOSE. The former is used to address multiple terminations at once, while the latter is used to indicate to a media gateway that it must select a termination satisfying the partially specified TerminationID. This allows, for instance, that a MGC instructs a MG to choose a circuit within a trunk group.

TerminationIDs can also be specified in a list. The use of TerminationIDList is recommended for cases where a hierarchical TerminationID structure is not possible and it is not desired to send individual commands for each TerminationID.

6.2.3 Packages

Different types of gateways may implement terminations that have widely differing characteristics. Variations in terminations are accommodated in the protocol by allowing terminations to have optional properties, events, signals and statistics implemented by MGs.

In order to achieve MG/MGC interoperability, such options are grouped into packages, and typically a termination realizes a set of such packages. More information on definition of packages can be found in clause 12. An MGC can audit a termination to determine which packages it realizes.

Properties, events, signals and statistics defined in packages, as well as parameters to them, are referenced by identifiers (IDs). Identifiers are scoped. For each package, PropertyIDs, EventIDs, SignalIDs, StatisticIDs and ParameterIDs have unique name spaces and the same identifier may be used in each of them. Two PropertyIDs in different packages may also have the same identifier, etc.

To support a particular package the MG must recognize all properties, signals, events and statistics defined in a package. It must also support all signal and event parameters. If the functionality behind these properties, signals, events and statistics is not implemented, the MG shall not return a syntax error or unknown ID error for any of these elements but should return error 501 "Not implemented".

The MG may support a subset of the values listed in a package for a particular property or parameter. If an unsupported value is specified by the MGC, the MG shall return error 501 "Not implemented".

When packages are extended, the properties, events, signals and statistics defined in the base package can be referred to using either the extended package name or the base package name. For example, if Package A defines Event e1, and Package B extends Package A, then B/e1 is an event for a termination implementing Package B. By definition, the MG must also implement the base package, but it is optional to publish the base package as an allowed interface. If it does publish A, then A would be reported on the Packages Descriptor in AuditValue as well as B, and Event A/e1 would be available on a termination. If the MG does not publish A, then only B/e1 would be available. If published through AuditValue, A/e1 and B/e1 are the same event.

For improved interoperability and backward compatibility, an MG may publish all packages supported by its terminations, including base packages from which extended packages are derived. An exception to this is in cases where the base packages are expressly defined as "Designed to be extended only".

6.2.4 Termination properties and descriptors

Terminations have properties. The properties have unique PropertyIDs. Most properties have default values, which are explicitly defined in this protocol specification or in a package (see clause 12) or set by provisioning. If not provisioned otherwise, the properties in all descriptors except TerminationState and LocalControl default to empty/"no value" when a termination is first created or returned to the NULL Context. When a termination is first created or returned to the NULL Context, this state represents an "idle" line, trunk or other entity. The default contents of the two exceptions are described in 7.1.5 and 7.1.7.

The provisioning of a property value in the MG will override any default value, be it supplied in this protocol specification or in a package. Therefore, if it is essential for the MGC to have full control over the property values of a termination, it should supply explicit values when adding the termination to a context. Alternatively, for a physical termination the MGC can determine any provisioned property values by auditing the termination while it is in the NULL Context.

There are a number of common properties for terminations and properties specific to media streams. The common properties are also called the TerminationState properties. For each media stream, there are local properties and properties of the received and transmitted flows.

Properties not included in the base protocol are defined in packages. These properties are referred to by a name consisting of the package name and a PropertyID. Most properties have default values described in the package description. Properties may be read-only or read/write. The possible values of a property may be audited, as can their current values. For properties that are read/write, the MGC can set their values. A property may be declared as "global" which has a single value shared by all terminations realizing the package. Related properties are grouped into descriptors for convenience.

When a termination is added to a context, the value of its read/write properties can be set by including the appropriate descriptors as parameters to the Add Command. Similarly, a property of a termination in a context may have its value changed by the Modify Command. Properties may also have their values changed when a termination is moved from one context to another as a result of a Move Command. In some cases, descriptors are returned as output from a command.

Setting properties on different terminations in the same context implicitly instructs the MG to perform certain functions. For example: if a G.711 codec is set on Termination A and a G.729 codec is set on Termination B, then the MG will activate a transcoding function as soon as a media flow is enabled between the two terminations (i.e., by setting the Mode Property to a state other than "Inactive" at each termination).

NOTE – To avoid unnecessary activation of MG resources, the MGC should set the Mode Property to "Inactive" for a given termination and stream until the properties to be used for that stream have been determined.

In general, if a descriptor is completely omitted from one of the aforementioned commands, the properties in that descriptor retain their prior values for the termination(s) upon which the command acts. On the other hand, if some read/write properties are omitted from a descriptor in a command (i.e., the descriptor is only partially specified), those properties will be reset to their default values for the termination(s) upon which the command acts, unless the package specifies other behaviour. For more details, see 7.1 dealing with the individual descriptors.

The above behaviour applies equally to signals and events and their respective parameters. The provisioning of Events Descriptors would include the RequestID to use and any parameters of the included events. Any provisioning in the MG with respect to Events Descriptors should be duplicated in the MGC to avoid error responses to Notify Commands from the MG.

The following table lists all of the possible descriptors and their use. Not all descriptors are legal as input or output parameters to every command.

Descriptor name	Description
Modem	Identifies modem type and properties when applicable. (Note)
Mux	Describes multiplex type for multimedia terminations (e.g., H.221, H.223, H.225.0) and terminations forming the input mux.
Media	A list of media stream specifications (see 7.1.4).
TerminationState	Properties of a termination (which can be defined in packages) that are not stream specific.
Stream	A list of Remote/Local/LocalControl Descriptors for a single stream.
Local	Contains properties that specify the media flows that the MG receives from the remote entity.
Remote	Contains properties that specify the media flows that the MG sends to the remote entity.
LocalControl	Contains properties (which can be defined in packages) that are of interest between the MG and the MGC.
Events	Describes events to be detected by the MG and what to do when an event is detected.
EventBuffer	Describes events to be detected by the MG when event buffering is active.
Signals	Describes signals (see 7.1.11) applied to terminations.
Audit	In Audit commands, identifies which information is desired.
Packages	In AuditValue, returns a list of packages realized by the termination.
DigitMap	Defines patterns against which sequences of a specified set of events are to be matched so they can be reported as a group rather than singly.
ServiceChange	In ServiceChange, what, why ServiceChange occurred, etc.
ObservedEvents	In Notify or AuditValue, report of events observed.
Statistics	In Subtract and Audit, report of statistics kept on a termination or stream.
Topology	Specifies flow directions between terminations in a context.
ContextAttribute	Contains properties (which can be defined in packages) that affect the context as a whole.
Error	Contains an Error Code and optionally error text; it may occur in command replies and in Notify requests.
NOTE – Modem Descriptor has been deprecated in ITU-T Rec. H.248.1 version 2 (05/2002).	

6.2.5 Root Termination

Occasionally, a command must refer to the gateway as an entity in itself, rather than a termination within it. A special TerminationID, "Root" is reserved for this purpose. Packages may be defined on Root. Root thus may have properties, events, signals and statistics. Accordingly, the Root TerminationID may appear in:

- a Modify Command – to change a property, send a signal or set an event;
- a Notify Command – to report an event;
- an AuditValue Command – to examine the values of properties and statistics implemented on Root;
- an AuditCapabilities Command – to determine what properties of Root are implemented;
- a ServiceChange – to declare the entire gateway in or out of service.

Any other use of the Root TerminationID is an error. Error Code 410 ("Incorrect identifier") shall be returned in these cases.

6.3 Wildcarding principles

This clause specifies the behaviour for wildcarding ContextIDs and TerminationIDs that shall be applied to all commands. In processing these commands two forms of wildcarding must be considered:

1) Context wildcarding;

2) Termination wildcarding.

For the purposes of the wildcarding procedures, a TerminationIDList consisting of more than one TerminationID is considered a wildcarded TerminationID. When executing a transaction that contains wildcarded contexts and optionally wildcarded terminations, all commands in the transaction are executed in order for a particular instance of ContextID before moving to a subsequent ContextID instance. In the case that there are multiple commands in a transaction, only when the TerminationID (wildcarded or specific) specified in the first command matches a specific instance of a ContextID are subsequent commands in the transaction executed. If a TerminationID (wildcarded or specific) of the subsequent command(s) in that transaction does not match the specific ContextID instance, then Error Code 431 ("No TerminationID matched a wildcard") is returned and processing of subsequent instances of the wildcard ContextID are stopped unless the command that generated the error is marked optional.

The execution of particular wildcard combinations is discussed below.

6.3.1 ContextID specific with TerminationID wildcarded

In the case where the ContextID is specific, when ALL is used in the TerminationID of a command, the effect is identical to repeating the command with each of the matching TerminationIDs. The use of ALL does not address the Root termination. Since each of these commands may generate a response, the size of the entire response may be large. Thus, if the wildcard matches more than one TerminationID in the context, all possible matches are attempted, with results reported for each one. If none of the terminations referenced by the wildcarded TerminationID are in the specific context, then Error Code 431 ("No TerminationID matched a wildcard") is returned. No errors are returned for individual terminations specified by the wildcarded TerminationID that are not in the specified context.

For example: Assume that a gateway has four terminations: t1/1, t1/2, t2/1 and t2/2. Assume that Context 1 has t1/1 and t2/1 in it and that Context 2 has t1/2 and t2/2 in it.

The command:

Context=1{Command=t1/*{Descriptor/s}}

returns:

Context=1{Command=t1/1{Descriptor/s}}

6.3.2 ContextID wildcarded (ALL) with TerminationID specific

In the case where the ContextID is wildcarded (i.e., ContextID = ALL) and the TerminationID is fully specified, the effect is identical to a command specifying the non-NULL context that contains the specified termination. Thus, a search must be made to find the context and only one instance of the command is executed. No errors are reported for contexts that do not contain the specified termination. If the termination is not contained in any (non-NULL) context, then Error Code 431 ("No TerminationID matched a wildcard") is returned. If there are no contexts other than NULL in existence, Error Code 411 ("The transaction refers to an unknown ContextID") is returned. Use of this form of action rather than one specifying the ContextID is discouraged but may be useful, for example in correcting conflicting state between MG and MGC.

For example: Taking the above gateway configuration, the command:

Context=*{Command=t1/1{Descriptor/s}}

returns:

Context=1{Command=t1/1{Descriptor/s}}

6.3.3 ContextID wildcarded (ALL) with TerminationID wildcarded

In the case where the ContextID is wildcarded (i.e., ContextID = ALL) and the TerminationID is wildcarded, the effect is identical to repeating the command with each of the TerminationIDs matching the wildcard for each non-NULL context that contains one or more of those matching TerminationIDs. Thus if the wildcard matches more than one TerminationID in the specific instance of the wildcarded ContextID, all possible matches are attempted, with results reported for each one. No errors are reported for contexts that do not contain a termination matching the wildcarded TerminationID. No errors are returned for individual terminations specified in the wildcarded TerminationID that are not in a specific instance of the wildcarded ContextID. If there are no matches to the wildcarded ContextID and TerminationID, then Error Code 431 ("No TerminationID matched a wildcard") is returned.

For example: Taking the above gateway configuration, the command:

Context=*{Command=t1/*{Descriptor/s}}

returns:

Context=1{Command=t1/1{Descriptor/s}}

Context=2 {Command=t1/2{Descriptor/s}}

In the case that multiple commands are contained in a wildcarded TerminationID and/or wildcarded ContextID request, then if the first command does not match the first ContextID and TerminationID instance, then the subsequent command in the request will not be executed for that instance.

6.3.4 Wildcarded responses

If individual responses are not required, a wildcard response may be requested. In such a case, a single response is generated, which contains the UNION of all of the individual responses which otherwise would have been generated, with duplicate values suppressed. For instance, given Termination Ta with Properties p1=a, p2=b and Termination Tb with Properties p2=c, p3=d, a UNION response would consist of a wildcarded TerminationID and the sequence of Properties p1=a, p2=b,c and p3=d. Wildcard responses may be particularly useful in the Audit commands. If a wildcard UNION response is used in conjunction with a wildcarded context, then a single response is sent with the UNION of all the individual terminations referenced by the TerminationID. The response would contain Context=ALL, a wildcarded TerminationID and the sequence of properties.

If an error occurs during the execution of a wildcarded request that specifies a wildcarded response, special handling is required to provide useful information about the error(s) while still maintaining a modest sized response. When a wildcarded response is requested, all instances (as specified above) of the command shall be executed even if one or more result in errors, but later commands in the transaction will not be executed (unless optional was specified). Multiple command responses shall be returned for the command that encountered the error. The first command response shall be the normal wildcard response containing the UNION of responses for those commands that succeeded. If none of them succeeded, the UNION shall be empty. Additional command responses for each TransactionID that failed shall be returned with the appropriate Error Descriptor.

For example, the command:

Context=*{Command=t1/*{Descriptor/s}}

Response to an error:

Context=*{Command=t1/*{Union response descriptors},

Command=t1/3{Error=errorcode}}

The encoding of the wildcarding mechanism is detailed in Annexes A and B.

7 Commands

The protocol provides commands for manipulating the logical entities of the protocol connection model, contexts and terminations. Commands provide control at the finest level of granularity supported by the protocol. For example, commands exist to add terminations to a context, modify terminations, subtract terminations from a context, and audit properties of contexts or terminations. Commands provide for complete control of the properties of contexts and terminations. This includes specifying which events a termination is to report, which signals/actions are to be applied to a termination and specifying the topology of a context (who hears/sees whom).

Most commands are for the specific use of the Media Gateway Controller as command initiator in controlling Media Gateways as command responders. The exceptions are the Notify and ServiceChange Commands: Notify is sent from Media Gateway to Media Gateway Controller, and ServiceChange may be sent by either entity. Below is an overview of the commands; they are explained in more detail in 7.2.

1) **Add**: The Add Command adds a termination to a context. The Add Command on the first termination in a context is used to create a context.
2) **Modify**: The Modify Command modifies the properties, events and signals of a termination.
3) **Subtract**: The Subtract Command disconnects a termination from its context and returns statistics on the termination's participation in the context. The Subtract Command on the last termination in a context deletes the context.
4) **Move**: The Move Command atomically moves a termination to another context.
5) **AuditValue**: The AuditValue Command returns the current state of properties, events, signals and statistics of terminations.
6) **AuditCapabilities**: The AuditCapabilities Command returns all the possible values for termination properties, events and signals allowed by the Media Gateway.
7) **Notify**: The Notify Command allows the Media Gateway to inform the Media Gateway Controller of the occurrence of events in the Media Gateway.
8) **ServiceChange**: The ServiceChange Command allows the Media Gateway to notify the Media Gateway Controller that a termination or group of terminations is about to be taken out of service or has just been returned to service. ServiceChange is also used by the MG to announce its availability to a MGC (registration), and to notify the MGC of impending or completed restart of the MG. The MGC may announce a handover to the MG by sending it a ServiceChange Command. The MGC may also use ServiceChange to instruct the MG to take a termination or group of terminations in or out of service.

These commands are detailed in 7.2.1 through 7.2.8.

7.1 Descriptors

The parameters to a command are termed descriptors. A descriptor consists of a name and a list of items. Some items may have values. Many commands share common descriptors. This subclause enumerates these descriptors. Descriptors may be returned as output from a command. In any such return of descriptor contents, an empty descriptor is represented by its name unaccompanied by any

list. Parameters and parameter usage specific to a given command type are described in the subclause that describes the command.

7.1.1 Specifying parameters

Command parameters are structured into a number of descriptors. In general, the text format of descriptors is DescriptorName=<someID>{parm=value, parm=value…}.

Parameters may be fully specified, overspecified or underspecified:

1) Fully specified parameters have a single, unambiguous value that the command initiator is instructing the command responder to use for the specified parameter.
2) Underspecified parameters, using the CHOOSE value, allow the command responder to choose any value it can support.
3) Overspecified parameters have a list of potential values. The list order specifies the command initiator's order of preference of selection. The command responder chooses one value from the offered list and returns that value to the command initiator.

If a required descriptor other than the Audit Descriptor is unspecified (i.e., entirely absent) from a command, the previous values set in that descriptor for that termination, if any, are retained. In commands other than Subtract, a missing Audit Descriptor is equivalent to an empty Audit Descriptor. The behaviour of the MG with respect to unspecified parameters within a descriptor varies with the descriptor concerned, as indicated in succeeding subclauses. Whenever a parameter is underspecified or overspecified, the descriptor containing the value chosen by the responder is included as output from the command.

Each command specifies the TerminationID the command operates on. This TerminationID may be "wildcarded". When the TerminationID of a command is wildcarded, the effect shall be as if the command was repeated with each of the TerminationIDs matched.

7.1.2 Modem Descriptor

The Modem Descriptor specifies the modem type and parameters, if any, required for use in e.g., H.324 and text conversation. The descriptor includes the following modem types: V.18, V.22, V.22 *bis*, V.32, V.32 *bis*, V.34, V.90, V.91, Synchronous ISDN, and allows for extensions. By default, no Modem Descriptor is present in a termination.

Use of the Modem Descriptor is deprecated in ITU-T Rec. H.248.1 Version 2 (05/2002) and subsequent versions. This means that the Modem Descriptor shall not be included as part of transmitted content and, if received, shall either be ignored or processed at the option of the implementation. Modem type is to be specified as an attribute of data streams in the Local Descriptor and the Remote Descriptor.

7.1.3 Multiplex Descriptor

In multimedia calls, a number of media streams are carried on a (possibly different) number of bearers. The Multiplex Descriptor associates the media and the bearers. The descriptor includes the multiplex type:

- H.221;
- H.223;
- H.226;
- V.76;
- N × 64K;
- possible extensions,

and a set of TerminationIDs representing the multiplexed bearers, in order. For example:

```
Mux = H.221{MyT3/1/2, MyT3/2/13, MyT3/3/6, MyT3/21/22}
```

The N × 64K multiplex type implements the N × 64K service (e.g., as defined by Q.931 Information Transfer Rate or Q.763 Transmission Medium Requirement). On the context side, it supports a single stream of wideband data. When a bearer termination is added implicitly to a context as a result of the creation of an N × 64K multiplexing termination, the Stream Descriptor for the bearer termination shall take on the same values as the Stream Descriptor defined for the multiplex termination, except that the bearer termination bandwidth shall be 64 kbit/s.

7.1.4 Media Descriptor

The Media Descriptor specifies the parameters for all the media streams. These parameters are structured into two descriptors: a TerminationState Descriptor, which specifies the properties of a termination that are not stream dependent, and one or more Stream Descriptors each of which describes a single media stream.

A stream is identified by a StreamID. The StreamID shall be in the range of 1 to 65535. The StreamID is used to link the streams in a context that belong together. Multiple streams exiting a termination shall be synchronized with each other. Within the Stream Descriptor, there are up to four subsidiary descriptors: LocalControl, Local, Remote and Statistics. The relationship between these descriptors is thus:

Media Descriptor

```
TerminationState Descriptor
Stream
Descriptor
   LocalControl Descriptor
   Local Descriptor
   Remote Descriptor
   Statistics Descriptor
```

As a convenience, the LocalControl, Local, Remote or Statistics Descriptors may be included in the Media Descriptor without an enclosing Stream Descriptor. In this case, the StreamID is assumed to be one.

7.1.5 TerminationState Descriptor

The TerminationState Descriptor contains the ServiceStates Property, the EventBufferControl Property and properties of a termination (defined in packages) that are not stream specific.

The ServiceStates Property describes the overall state of the termination (not stream specific). A termination can be in one of the following states: "Test", "OutOfService", or "InService". The "Test" state indicates that the termination is being tested. The state "OutOfService" indicates that the termination cannot be used for traffic. The state "InService" indicates that a termination can be used or is being used for normal traffic. "InService" is the default state.

Values assigned to properties may be simple values (integer/string/enumeration) or may be underspecified, where more than one value is supplied and the MG may make a choice:

- Alternative Values: multiple values in a list, one of which must be selected;
- Ranges: minimum and maximum values, a value between min and max must be selected, boundary values included;
- Greater Than/Less Than: value must be greater/less than specified value;
- CHOOSE Wildcard: the MG chooses from the allowed values for the property.

The EventBufferControl Property specifies whether events are buffered following detection of an event in the Events Descriptor, or processed immediately. See 7.1.9 for details.

7.1.5.1 TerminationState Properties

7.1.5.1.1 ServiceStates

Property Name: ServiceStates

Description: The value of this property indicates the current service state of the termination.

Type: Enumeration

Possible values:

InService:	The termination is in-service and functioning normally.
OutOfService:	The termination is out-of-service and not available for traffic.
Test:	The termination is undergoing testing.

Default: InService

Defined in: TerminationState

Characteristics: Read/Write

7.1.5.1.2 EventBufferControl

Property Name: EventBufferControl

Description: Specifies whether events are buffered following detection of an event in the Events Descriptor, or processed immediately. See 7.1.9.

Type: Enumeration

Possible values:

LockStep:	Events are buffered and processed per 7.1.9.
OFF:	Events are processed immediately.

Default: OFF

Defined in: TerminationState

Characteristics: Read/Write

7.1.6 Stream Descriptor

A Stream Descriptor specifies the parameters of a single bidirectional stream. These parameters are structured into three descriptors: one that contains termination properties specific to a stream and one each for local and remote flows. The Stream Descriptor includes a StreamID which identifies the stream. Streams are created by specifying a new StreamID on one of the terminations in a context. A stream is deleted by setting empty Local and Remote Descriptors for the stream with ReserveGroup and ReserveValue in LocalControl set to "False" on all terminations in the context that previously supported that stream.

StreamIDs are of local significance between MGC and MG and they are assigned by the MGC. Within a context, StreamID is a means by which to indicate which media flows are interconnected: streams with the same StreamID are connected.

If a termination is moved from one context to another, the effect on the context to which the termination is moved is the same as in the case that a new termination were added with the same StreamIDs as the moved termination.

7.1.7 LocalControl Descriptor

The LocalControl Descriptor contains the Mode Property, the ReserveGroup and ReserveValue Properties and properties of a termination (defined in packages) that are stream specific, and are of interest between the MG and the MGC. Values of properties may be specified as in 7.1.1.

The allowed values for the Mode Property are "SendOnly", "RecvOnly", "SendRecv", "Inactive" and "LoopBack". "SendOnly", "RecvOnly" and "LoopBack" are with respect to the exterior of the context, so that, for example, a stream set to mode = "SendOnly" does not pass received media into the context. When a stream is set to "LoopBack" on a termination, media received (Local Descriptor) on the termination will be looped back to the sending side (Remote Descriptor) of the termination and no media is passed between that termination and other terminations in the context. The looped back media shall be sent according to the Remote Descriptor. The default value for the Mode Property is "Inactive". Signals and events are not affected by the Mode Property. The LocalControl Mode Property takes precedence over any mode specified in the Local and Remote Descriptors.

The boolean-valued Reserve Properties, ReserveValue and ReserveGroup, of a termination indicate what the MG is expected to do when it receives a Local and/or Remote Descriptor.

If the value of a Reserve Property is "True", the MG shall reserve resources for all alternatives specified in the Local and/or Remote Descriptors for which it currently has resources available. It shall respond with the alternatives for which it reserves resources. If it cannot support any of the alternatives, it shall respond with a reply to the MGC that contains empty Local and/or Remote Descriptors. If media begins to flow while more than a single alternative is reserved, media packets may be sent or received on any of the alternatives and must be processed, although only a single alternative may be active at any given time.

If the value of a Reserve Property is False, the MG shall choose one of the alternatives specified in the Local Descriptor (if present) and one of the alternatives specified in the Remote Descriptor (if present). If the MG has not yet reserved resources to support the selected alternative, it shall reserve the resources. If, on the other hand, it already reserved resources for the termination addressed (because of a prior exchange with ReserveValue and/or ReserveGroup equal to "True"), it shall release any excess resources it reserved previously. Finally, the MG shall send a reply to the MGC containing the alternatives for the Local and/or Remote Descriptor that it selected. If the MG does not have sufficient resources to support any of the alternatives specified, it shall respond with Error Code 510 ("Insufficient resources").

The default value of ReserveValue and ReserveGroup is "False". More information on the use of the two Reserve Properties is provided in 7.1.8.

A new setting of the LocalControl Descriptor completely replaces the previous setting of that descriptor in the MG. Thus, to retain information from the previous setting, the MGC must include that information in the new setting. If the MGC wishes to delete some information from the existing descriptor, it merely resends the descriptor (in a Modify Command) with the unwanted information stripped out.

NOTE – The Mode Property is also known as "StreamMode" in the encoding definitions in Annexes A and B. These terms are interchangeable within H.248.

7.1.7.1 LocalControl Properties

7.1.7.1.1 Mode

Property Name: StreamMode

Description: The value of this property indicates the current service state of the termination.

Type: Enumeration

Possible values:

Inactive:	The termination does not pass any media for the stream.
SendOnly:	The termination passes media for the stream from the interior to the exterior of the context.
RecvOnly:	The termination passes media for the stream from the exterior to the interior of the context.
SendRecv:	The termination passes media for the stream both into and out of the context.
LoopBack:	The termination loops received media for the stream back to the sender.

Default: Inactive

Defined in: LocalControl

Characteristics: Read/Write

7.1.7.1.2 ReserveGroup

Property Name: ReserveGroup

Description: Specifies whether the MG should reserve the resources to support a single media group or as many media groups as it can, as they are defined in the Local and Remote Descriptors. See 7.1.8.

NOTE – The term "media group" designates the content of the ASN.1 production "PropertyGroup" (see Annex A) in the binary encoding, or an individual SDP session description in the text encoding.

Type: Boolean

Possible values:

True:	The MG is to reserve all possible media groups indicated in the Local and Remote Descriptors.
False:	The MG is to reserve a single media group from each of the Local and the Remote Descriptors according to the policies described in 7.1.8.

Default: False

Defined in: LocalControl

Characteristics: Read/Write

7.1.7.1.3 ReserveValue

Property Name: ReserveValue

Description: Specifies whether the MG should reserve the resources to support a single set of property values (e.g., a single codec and its associated attributes) or as many such sets as it can, as they are defined in the Local and Remote Descriptors. See 7.1.8.

Type: Boolean

Possible values:

True:	The MG is to reserve resources to serve as many as possible of the sets of property values indicated in the selected media group (if ReserveGroup is False) or in each media group (if ReserveGroup is True) in the Local and Remote Descriptors.

False: The MG is to reserve a single set of property values from those indicated in the selected media group (if ReserveGroup is False) or in each media group (if ReserveGroup is True) in the Local and Remote Descriptors.

Default: False

Defined in: LocalControl

Characteristics: Read/Write

7.1.8 Local and Remote Descriptors

The MGC uses Local and Remote Descriptors to reserve and commit MG resources for media decoding and encoding for the given stream(s) and termination to which they apply. The MG includes these descriptors in its response to indicate what it is actually prepared to support. The MG shall include additional properties and their values in its response if these properties are mandatory yet not present in the requests made by the MGC (e.g., by specifying detailed video encoding parameters where the MGC only specified the payload type).

To avoid ambiguity when requesting the MG to reserve and commit resources, the MGC should supply as much information as needed when using underspecification (i.e., CHOOSE) so that the MG can make an unambiguous selection. For example when using CHOOSE without specifying the required application type (e.g., "media name" in case of SDP encoding), further information may be needed (e.g., attribute lines in case of SDP encoding).

Local refers to the media received by the MG and Remote refers to the media sent by the MG.

When text encoding the protocol, the descriptors consist of session descriptions as defined in SDP (RFC 2327). In session descriptions sent from the MGC to the MG, the following exceptions to the syntax of RFC 2327 are allowed:

- the "s = ", "t = " and "o = " lines are optional;
- the use of CHOOSE is allowed in place of a single parameter value; and
- the use of alternatives is allowed in place of a single parameter value.

A Stream Descriptor specifies a single bidirectional media stream and so a single session description must not include more than one media description ("m = " line). A Stream Descriptor may contain additional session descriptions as alternatives. Each media stream for a termination must appear in distinct Stream Descriptors. When multiple session descriptions are provided in one descriptor, the "v = " lines are required as delimiters; otherwise they are optional in session descriptions sent to the MG. Implementations shall accept session descriptions that are fully conformant to RFC 2327 according to the above restrictions. When binary encoding the protocol, the descriptor consists of groups of properties (tag-value pairs) as specified in Annex C. Each such group may contain the parameters of a session description.

Below, the semantics of the Local and Remote Descriptors are specified in detail. The specification consists of two parts. The first part specifies the interpretation of the contents of the descriptor. The second part specifies the actions the MG must take upon receiving the Local and Remote Descriptors. The actions to be taken by the MG depend on the values of the ReserveValue and ReserveGroup Properties of the LocalControl Descriptor.

Either the Local or the Remote Descriptor or both may be:

- unspecified (i.e., absent);
- empty;
- underspecified through use of CHOOSE in a property value;
- fully specified; or

- overspecified through presentation of multiple groups of properties and possibly multiple property values in one or more of these groups.

Where the descriptors have been passed from the MGC to the MG, they are interpreted according to the rules given in 7.1.1, with the following additional comments for clarification:

a) An unspecified Local or Remote Descriptor is considered to be a missing mandatory parameter. It requires the MG to use whatever was last specified for that descriptor. It is possible that there was no previously specified value, in which case the descriptor concerned is ignored in further processing of the command.

b) An empty Local (Remote) Descriptor in a message from the MGC signifies a request to release any resources reserved for the media flow received (sent).

c) If multiple groups of properties are present in a Local or Remote Descriptor or multiple values within a group, the order of preference is descending.

d) Underspecified or overspecified properties within a group of properties sent by the MGC are requests for the MG to choose one or more values which it can support for each of those properties. In case of an overspecified property, the list of values is in descending order of preference.

Subject to the above rules, subsequent action depends on the values of the ReserveValue and ReserveGroup Properties in LocalControl.

If ReserveGroup is "True", the MG reserves the resources required to support as many as possible of the requested property group alternatives that it can currently support. If ReserveValue is "True", the MG reserves the resources required to support as many as possible of the requested property value alternatives that it can currently support.

NOTE – If a Local or Remote Descriptor contains multiple groups of properties, and ReserveGroup is "True", then the MG is requested to reserve resources so that it can decode or encode the media stream according to any of the alternatives. For instance, if the Local Descriptor contains two groups of properties, one specifying packetized G.711 A-law audio and the other G.723.1 audio, the MG reserves resources so that it can decode one audio stream encoded in either G.711 A-law format or G.723.1 format. The MG does not have to reserve resources to decode two audio streams simultaneously, one encoded in G.711 A-law and one in G.723.1. The intention for the use of ReserveValue is analogous.

If ReserveGroup is "True" or ReserveValue is "True", then the following rules apply:

- If the MG has insufficient resources to support all alternatives requested by the MGC, and the MGC requested resources in both Local and Remote, the MG should reserve resources to support at least one alternative each within Local and Remote.
- If the MG has insufficient resources to support at least one alternative within a Local (Remote) Descriptor received from the MGC, it shall return an empty Local (Remote) in response.
- In its response to the MGC, when the MGC included Local and Remote Descriptors, the MG shall include Local and Remote Descriptors for all groups of properties and property values it reserved resources for. If the MG is incapable of supporting at least one of the alternatives within the Local (Remote) Descriptor received from the MGC, it shall return an empty Local (Remote) Descriptor.
- If the Mode Property of the LocalControl Descriptor is "RecvOnly", "SendRecv", or "LoopBack", the MG must be prepared to receive media encoded according to any of the alternatives included in its response to the MGC.

If ReserveGroup is "False" and ReserveValue is "False", then the MG should apply the following rules to resolve Local and Remote to a single alternative each:

- The MG chooses the first alternative in Local for which it is able to support at least one alternative in Remote.

- If the MG is unable to support at least one Local and one Remote alternative, it returns Error Code 510 ("Insufficient resources").
- The MG returns its selected alternative in each of Local and Remote.

A new setting of a Local or Remote Descriptor completely replaces the previous setting of that descriptor in the MG. Thus, to retain information from the previous setting, the MGC must include that information in the new setting. If the MGC wishes to delete some information from the existing descriptor, it merely resends the descriptor (in a Modify Command) with the unwanted information stripped out.

7.1.9 Events Descriptor

The Events Descriptor parameter contains a RequestID and a list of events that the Media Gateway is requested to detect and report. The RequestID is used to correlate the request with the notifications that it may trigger. Requested events include, for example, fax tones, continuity test results, and on-hook and off-hook transitions. The RequestID is omitted if the Events Descriptor is empty (i.e., no events are specified).

Each event in the descriptor contains the event name, an optional StreamID, an optional KeepActive flag, an optional NotifyBehaviour flag, an optional ResetEventsDescriptor flag and optional parameters. The event name consists of a PackageID (where the event is defined) and an EventID. The ALL wildcard may be used for the EventID, indicating that all events from the specified package have to be detected. The default StreamID is 0, indicating that the event to be detected is not related to a particular media stream. Events can have parameters. This allows a single event description to have some variation in meaning without creating large numbers of individual events. Further event parameters are defined in the package.

If a DigitMap completion event is present or implied in the Events Descriptor, the EventDM Parameter is used to carry either the name or the value of the associated DigitMap. See 7.1.14 for further details.

When an event is processed against the contents of an active Events Descriptor and found to be present in that descriptor ("recognized"), the default action of the MG is to send a Notify Command to the MGC. Notification may be deferred if the event is absorbed into the current dial string of an active DigitMap (see 7.1.14). The sending of a Notify Command may be influenced by the NotifyBehaviour flag. Moreover, event recognition may cause currently active signals to stop, or may cause the current Events and/or Signals Descriptor to be replaced, as described at the end of this subclause. Unless the Events Descriptor is replaced by another Events Descriptor, it remains active after an event has been recognized.

If the value of the EventBufferControl Property equals "LockStep", following detection of such an event, normal handling of events is suspended. Any event which is subsequently detected and occurs in the EventBuffer Descriptor is added to the end of the EventBuffer (a FIFO queue), along with the time that it was detected. The MG shall wait for a new Events Descriptor to be loaded. A new Events Descriptor can be loaded either as the result of receiving a command with a new Events Descriptor, or by activating an embedded Events Descriptor.

If EventBufferControl equals "Off", the MG continues processing based on the active Events Descriptor.

In the case of an embedded Events Descriptor being activated, the MG continues event processing based on the newly activated Events Descriptor.

NOTE 1 – For purposes of EventBuffer handling, activation of an embedded Events Descriptor is equivalent to receipt of a new Events Descriptor.

When the MG receives a command with a new Events Descriptor, one or more events may have been buffered in the EventBuffer in the MG. The value of EventBufferControl, then determines how the MG treats such buffered events.

Case 1

If EventBufferControl equals "LockStep" and the MG receives a new Events Descriptor, it will check the FIFO EventBuffer and take the following actions:

1) If the EventBuffer is empty, the MG waits for detection of events based on the new Events Descriptor.

2) If the EventBuffer is non-empty, the MG processes the FIFO queue starting with the first event:

   a) If the event in the queue is in the events listed in the new Events Descriptor, the MG acts on the event and removes the event from the EventBuffer. The time stamp of the Notify shall be the time the event was actually detected. The MG then waits for a new Events Descriptor. While waiting for a new Events Descriptor, any events detected that appear in the EventBuffer Descriptor will be placed in the EventBuffer. When a new Events Descriptor is received, the event processing will repeat from step 1).

   b) If the event is not in the new Events Descriptor, the MG shall discard the event and repeat from step 1).

Case 2

If EventBufferControl equals "Off" and the MG receives a new Events Descriptor, it processes new events with the new Events Descriptor.

If the MG receives a command instructing it to set the value of EventBufferControl to "Off", all events in the EventBuffer shall be discarded.

The MG may report several events in a single transaction as long as this does not unnecessarily delay the reporting of individual events.

For procedures regarding transmitting the Notify Command, refer to the appropriate annex or H.248.x Recommendation for specific transport considerations.

The default value of EventBufferControl is "Off".

NOTE 2 – Since the EventBufferControl Property is in the TerminationState Descriptor, the MG might receive a command that changes the EventBufferControl Property and does not include an Events Descriptor.

Normally, recognition of an event shall cause any active signals to stop. When KeepActive is specified in the event, the MG shall not interrupt any signals active on the termination on which the event is detected.

The NotifyBehaviour flag may be used to indicate that the Notify Command is:

- sent immediately (NotifyImmediate: this is the default);
- never sent (NeverNotify);
- or regulated (i.e., sent or suppressed) according to the MGC load (RegulatedNotify).

See E.15 for more details on the use of NotifyBehaviour. When used with DigitMaps the notify behaviour occurs when the active DigitMap is completed. The Regulated Notify may have an alternate regulated embedded Events or Signals Descriptor associated with it. If a notification is regulated (i.e., suppressed), then this alternate regulated embedded descriptor shall be activated. If the Notify is not regulated, then the original embedded descriptor is triggered. If NotifyImmediate or NeverNotify is set, then any original embedded descriptor is triggered on detection of the event.

The result of encountering the ResetEventDescriptor flag set against an event depends on whether or not the Events Descriptor containing the event is an embedded Events Descriptor. In the case of

an embedded Events Descriptor the ResetEventsDescriptor flag causes the Events Descriptor for that termination to be reset to its state prior to when the embedded descriptor was activated (i.e., the last Events Descriptor explicitly set by a Modify Command or, if there has been no Modify Command since the termination was last reset by being put into the NULL Context, the MG-provisioned Events Descriptor). In the case of a non-embedded Events Descriptor the ResetEventsDescriptor flag causes the active Events Descriptor for that termination to be reset by re-activating any DigitMap completion event(s) that have been matched and deactivated.

The ResetEventsDescriptor flag shall be set on physical terminations only. The ResetEventsDescriptor flag shall be acted on when the termination is in the NULL Context. If the ResetEventsDescriptor flag is encountered when the termination is out of the NULL Context, then it shall have no effect.

An event can include an embedded Signals Descriptor and/or an embedded Events Descriptor which, if present, replaces the current Signals/Events Descriptor when the event is recognized. It is possible, for example, to specify that the dial-tone signal be generated when an off-hook event is recognized, or that the dial-tone signal be stopped when a digit is recognized. A media gateway controller shall not send Events Descriptors with an event both marked KeepActive and containing an embedded Signals Descriptor.

Only one level of embedding is permitted. An embedded Events Descriptor shall not contain another embedded Events Descriptor; an embedded Events Descriptor may contain an embedded Signals Descriptor.

An Events Descriptor received by a media gateway replaces any previous Events Descriptor. Event notification in process shall complete, and events detected after the command containing the new Events Descriptor executes, shall be processed according to the new Events Descriptor.

An empty Events Descriptor disables all event recognition and reporting. An empty EventBuffer Descriptor clears the EventBuffer and disables all event accumulation in "LockStep" mode: the only events reported will be those occurring while an Events Descriptor is active. If an empty Events Descriptor is activated while the termination is operating in "LockStep" mode, the EventBuffer is immediately cleared.

7.1.10 EventBuffer Descriptor

The EventBuffer Descriptor contains a list of events, with their parameters if any, that the MG is requested to detect and buffer when EventBufferControl equals LockStep (see 7.1.9).

7.1.11 Signals Descriptor

Signals are MG-generated media such as tones and announcements as well as bearer-related signals such as hookswitch. More complex signals may include a sequence of such simple signals interspersed with and conditioned upon the receipt and analysis of media or bearer-related signals. Examples include echoing of received data as in the Continuity Test Package. Signals may also request preparation of media content for future signals.

A Signals Descriptor is a parameter that contains the set of signals that the Media Gateway is asked to apply to a termination. A Signals Descriptor contains a number of signals and/or sequential signal lists. A Signals Descriptor may contain zero signals and sequential signal lists. Support of sequential signal lists is optional.

Signals are defined in packages. Signals shall be named with a PackageID (in which the signal is defined) and a SignalID. No wildcard shall be used in the SignalID. Signals that occur in a Signals Descriptor have an optional StreamID parameter (default is 0, to indicate that the signal is not related to a particular media stream), an optional signal type (see below), an optional duration and possibly parameters defined in the package that defines the signal. This allows a single signal to have some variation in meaning, obviating the need to create large numbers of individual signals.

The optional NotifyCompletion Parameter allows a MGC to indicate that it wishes to be notified when the signal finishes playout. The possible cases are that the signal timed out (or otherwise completed on its own), that it was interrupted by an event, that it completed a cycle/iteration of the signal, that it was halted when a Signals Descriptor was replaced, or that it stopped or never started for other reasons. If the NotifyCompletion Parameter is not included in a Signals Descriptor, notification is generated only if the signal stopped or was never started for other reasons. For reporting to occur, the Signal Completion Event (see E.1.2) must be enabled in the currently active Events Descriptor. The optional parameter "RequestID" may be associated with a particular instance of a SignalID where multiple signals of the same SignalID are requested. This allows the MGC to distinguish between different Signal Completion ObservedEvents for that particular instance of a signal. The RequestID parameter shall not be included if the NotifyCompletion Parameter is not present.

The duration is an integer value that is expressed in hundredths of a second.

There are three types of signals:

- OnOff (OO): the signal lasts until it is turned off;
- TimeOut (TO): the signal lasts until it is turned off or a specific period of time elapses;
- Brief (BR): the signal will stop on its own unless a new Signals Descriptor is applied that causes it to stop; no timeout value is needed.

If a signal of default type other than TO has its type overridden to type TO in the Signals Descriptor, the duration parameter must be present.

If the signal type is specified in a Signals Descriptor, it overrides the default signal type (see 12.1.4). If duration is specified for an on/off signal, it shall be ignored.

A sequential signal list consists of a SignalListID and a sequence of signals to be played sequentially. Only the trailing element of the sequence of signals in a sequential signal list may be an on/off signal. The duration of a sequential signal list is the sum of the durations of the signals it contains plus the sum of the intersignal delay timings specified as parameters to the signals.

If an intersignal delay is specified for a signal that is not in a sequential signal list or is the last element in a sequential signal list, it shall be ignored and the signal shall be determined to have completed at the termination of the signal, prior to the application of the intersignal delay timing. The duration of a signal in a SignalList with an intersignal delay includes the intersignal delay timing.

Multiple signals and sequential signal lists in the same Signals Descriptor shall be played simultaneously.

Signals are defined as proceeding from the termination towards the exterior of the context unless otherwise specified. If the signal direction is specified in a Signals Descriptor, it overrides the default signal direction. For those signals which have a direction parameter, the base protocol direction parameter takes precedence over any package-defined direction parameter when both are specified. If the MGC specifies a direction for a signal with which the MG cannot comply, the MG shall return Error Code 501 ("Not implemented"). When the same signal is applied to multiple terminations within one transaction, the MG should consider using the same resource to generate these signals.

Production of a signal on a termination is stopped by application of a new Signals Descriptor, or detection of an event on the termination (see 7.1.9).

A new Signals Descriptor replaces any existing Signals Descriptor. Any signals applied to the termination not in the replacement descriptor shall be stopped, and new signals are applied, except as follows. Signals present in the replacement descriptor and containing the KeepActive flag shall be continued if they are currently playing and have not already completed. If a replacement Signals

Descriptor contains a signal that is not currently playing and contains the KeepActive flag, that signal shall be ignored. If the replacement descriptor contains a sequential signal list with the same identifier as the existing descriptor, then:

- the signal type and sequence of signals in the sequential signal list in the replacement descriptor shall be ignored; and
- the playing of the signals in the sequential signal list in the existing descriptor shall not be interrupted.

7.1.12 Audit Descriptor

The Audit Descriptor specifies what information is to be audited. The Audit Descriptor specifies the list of descriptors and/or individual properties to be returned. Audit may be used in any command to force the return of any descriptor containing the current values of its properties, events, signals and statistics even if that descriptor was not present in the command, or had no underspecified parameters. Possible items in the Audit Descriptor are:

Modem (Deprecated, see 7.1.2)	
Mux	
Events	
Media	
Signals	
ObservedEvents	
DigitMap	
Statistics	
Packages	
EventBuffer	
Individual Audit Items:	
Media Properties Events EventBuffer Signals, SignalLists	DigitMaps Statistics Packages ContextAttribute

The Audit Descriptor may be empty, in which case, no descriptors are returned. This is useful in the Subtract Command to inhibit return of statistics, especially when using wildcard.

7.1.13 ServiceChange Descriptor

The ServiceChange Descriptor contains the following parameters:

- ServiceChangeMethod;
- ServiceChangeReason;
- ServiceChangeAddress;
- ServiceChangeDelay;
- ServiceChangeProfile;
- ServiceChangeVersion;
- ServiceChangeMGCID;
- TimeStamp;
- Extension;
- ServiceChangeInfo;

- ServiceChangeIncompleteFlag.

See 7.2.8.

7.1.14 DigitMap Descriptor

7.1.14.1 DigitMap definition, creation, modification and deletion

A DigitMap is a dialing plan resident in the Media Gateway used for detecting and reporting digit events received on a termination. The DigitMap Descriptor contains a DigitMap name and the DigitMap to be assigned. A DigitMap may be preloaded into the MG by management action and referenced by name in an Events Descriptor, may be defined dynamically and subsequently referenced by name, or the actual digitmap itself may be specified in the Events Descriptor. It is permissible for a DigitMap completion event within an Events Descriptor to refer by name to a DigitMap which is defined by a DigitMap Descriptor within the same command, regardless of the transmitted order of the respective descriptors.

DigitMaps defined in a DigitMap Descriptor can occur in any of the standard termination manipulation commands of the protocol. A DigitMap, once defined, can be used on all terminations specified by the (possibly wildcarded) TerminationID in such a command. DigitMaps defined on the Root Termination are global and can be used on every termination in the MG, provided that a DigitMap with the same name has not been defined on the given termination. When a DigitMap is defined dynamically in a DigitMap Descriptor:

- A new DigitMap is created by specifying a name that is not yet defined. The value shall be present.
- A DigitMap value is updated by supplying a new value for a name that is already defined. Terminations presently using the DigitMap shall continue to use the old definition; subsequent Events Descriptors specifying the name, including any Events Descriptor in the command containing the DigitMap Descriptor, shall use the new one.
- A DigitMap is deleted by supplying an empty value for a name that is already defined. Terminations presently using the DigitMap shall continue to use the old definition.

7.1.14.2 DigitMap timers

The collection of digits according to a DigitMap may be protected by three timers, viz. a start timer (T), short timer (S), and long timer (L).

1) The start timer (T) is used prior to any digits being available for processing against the DigitMap. If the start timer is overridden with the value set to zero (T = 0), then the start timer shall be disabled. This implies that the MG will wait indefinitely for digits.
2) If the Media Gateway can determine that at least one more digit is needed for a digit string to match any of the allowed patterns in the DigitMap, then the interdigit timer value should be set to a long (L) duration (e.g., 16 seconds).
3) If the digit string has matched one of the patterns in a DigitMap, but it is possible that more digits could be received which would cause a match with a different pattern, then instead of reporting the match immediately, the MG must apply the short timer (S) and wait for more digits.

The timers are configurable parameters to a DigitMap. Default values of these timers should be provisioned on the MG, but can be overridden by values specified within the DigitMap.

7.1.14.3 DigitMap syntax

The formal syntax of the DigitMap is described by the DigitMap rule in the formal syntax description of the protocol (see Annexes A and B). A DigitMap, according to this syntax, is defined either by a string or by a list of strings. Each string in the list is an alternative event sequence, specified either as a sequence of DigitMap symbols or as a regular expression of DigitMap symbols.

These DigitMap symbols, the digits "0" through "9" and letters "A" through a maximum value depending on the signalling system concerned, but never exceeding "K", correspond to specified events within a package which has been designated in the Events Descriptor on the termination to which the DigitMap is being applied. (The mapping between events and DigitMap symbols is defined in the documentation for packages associated with channel-associated signalling systems such as DTMF, MF, or R2. Digits "0" through "9" must be mapped to the corresponding digit events within the signalling system concerned. Letters should be allocated in logical fashion, facilitating the use of range notation for alternative events.)

The letter "x" is used as a wildcard, designating any event corresponding to symbols in the range "0"-"9". The string may also contain explicit ranges and, more generally, explicit sets of symbols, designating alternative events any one of which satisfies that position of the DigitMap. Finally, the dot symbol "." stands for zero or more repetitions of the event selector (event, range of events, set of alternative events, or wildcard) that precedes it. As a consequence of the third timing rule above, inter-event timing while matching a terminal dot symbol uses the short timer by default.

In addition to these event symbols, the string may contain "S" and "L" inter-event timing specifiers and the "Z" duration modifier. "S" and "L" respectively indicate that the MG should use the short (S) timer or the long (L) timer for subsequent events, overriding the timing rules described above. If an explicit timing specifier is in effect in one alternative event sequence, but none is given in any other candidate alternative, the timer value set by the explicit timing specifier must be used. If all sequences with explicit timing controls are dropped from the candidate set, timing reverts to the default rules given above. If used inside a range notation, the S and L specifiers shall be ignored. Finally, if conflicting timing specifiers are in effect in different alternative sequences, the long timer shall be used.

A "Z" designates a long duration event: placed in front of the symbol(s) designating the event(s) which satisfy a given digit position, it indicates that that position is satisfied only if the duration of the event exceeds the long-duration threshold. The value of this threshold is assumed to be provisioned in the MG, but, like the T, L, and S timers, can be overridden by specification within the DigitMap. If the Z specifier is not followed by a digit (0-9 or A-K), then the MG shall reject the digitmap as invalid procedure. When used in a range notation, the Z specifier applies solely to the immediately following digit. When used immediately prior to a range, the Z modifier applies to all digits in the range (thereby requiring a match in the range to be long duration).

7.1.14.4 DigitMap completion event

A DigitMap is active while the Events Descriptor which invoked it is active and it has not completed. A DigitMap completes when:

- a timer has expired; or
- an alternative event sequence has been matched and no other alternative event sequence in the DigitMap could be matched through detection of an additional event (unambiguous match); or
- an event has been detected such that a match to a complete alternative event sequence of the DigitMap will be impossible no matter what additional events are received.

Upon completion, a DigitMap completion event, as defined in the package providing the events being mapped into the DigitMap, shall be generated. At that point, the DigitMap is deactivated. Subsequent events in the package are processed as follows:

- If EventBufferControl is "LockStep", subsequent digit events are processed in the same way as any other events;
- If EventBufferControl is "OFF", if the active Events Descriptor did not change, and if individual digit events are not enabled within that descriptor, then digit buffering will be

initiated. Buffering will continue until the buffering time specified in the original DigitMap completion event has expired or until the active Events Descriptor is replaced.

The digit buffer shall take the logical form of a dial string which includes the digit characters as represented in the DigitMap, possibly preceded by 'Z'. The threshold value of tone duration used to identify long events shall be the same as that used with the most recently completed DigitMap.

Buffering time defaults to zero (no buffering) unless explicitly set otherwise within the DigitMap completion event. If buffering ceases due to buffering timer expiry, the contents of the digit buffer are discarded.

If buffering was stopped by a new Events Descriptor and that Events Descriptor contains a new DigitMap completion event from the same package as the previous one, any buffered digits are processed against the DigitMap as described below. Buffered digits not consumed by the new DigitMap are handled as if they had been observed after that map completed.

If, instead, the new Events Descriptor enables the reporting of individual digit events, the entire set of buffered digits shall immediately be processed, the applicable events reported, and the buffer cleared.

Finally, if the new Events Descriptor enables neither a DigitMap completion event nor the reporting of individual digit events from the package concerned, the buffer contents are discarded and buffering is terminated.

7.1.14.5 DigitMap procedures

Pending completion, successive events shall be processed according to the following rules:

1) The "current dial string", an internal variable, is initially empty. The set of candidate alternative event sequences includes all of the alternatives specified in the DigitMap.

2) At each step, if buffered digits are available, the oldest one (with possible accompanying long duration (Z) qualifier) is removed from the buffer and processing moves to the next step as if the digit event had just been observed. Otherwise a timer is set to wait for the next event, based either on the default timing rules given above, or on explicit timing specified in one or more alternative event sequences. If the timer expires and a member of the candidate set of alternatives is fully satisfied, a timeout completion with full match is reported. If the timer expires and part or none of any candidate alternative is satisfied, a timeout completion with partial match is reported. In either case, if the DigitMap completion event allows for detailed timeout reporting, the reported dial string will end with 'L', 'S', or 'T' as appropriate.

3) If an event is detected before the timer expires, it is mapped to a digit string symbol and provisionally added to the end of the current dial string. The duration of the event (long or not long) is noted if, and only if, this is relevant in the current symbol position (because at least one of the candidate alternative event sequences includes the "Z" modifier at this position in the sequence).

4) The current dial string is compared to the candidate alternative event sequences. If, and only if, a sequence expecting a long-duration event at this position is matched (i.e., the event had long duration and met the specification for this position), then any alternative event sequences not specifying a long duration event at this position are discarded, and the current dial string is modified by inserting a "Z" in front of the symbol representing the latest event. Any sequence expecting a long-duration event at this position, but not matching the observed event, is discarded from the candidate set. If alternative event sequences not specifying a long duration event in the given position remain in the candidate set after application of the above rules, the observed event duration is treated as irrelevant in assessing matches to them.

5) If exactly one candidate remains and it has been fully matched, a completion event is generated indicating an unambiguous match. If no candidates remain, the latest event is removed from the current dial string and a completion event is generated indicating full match if one of the candidates from the previous step was fully satisfied before the latest event was detected or partial match otherwise. The event removed from the current dial string will then be reported as a separate event, buffered, or discarded according to the rules described in the previous clause. This statement is qualified as follows:

   a) The DigitMap completion event may specify that the removed event be reported as a parameter of the completion event. This occurs independently of subsequent processing of the digit event.

   b) The DigitMap completion event may specify that the extra digit should be discarded. In this case, it is discarded immediately. Any buffering or other processing applies only to subsequent events.

6) If no completion event is reported out of step 5), processing returns to step 2).

7.1.14.6 DigitMap activation

A DigitMap is activated whenever a new Events Descriptor is applied to the termination or embedded Events Descriptor is activated, and that Events Descriptor contains a DigitMap completion event. The DigitMap completion event contains an EventDM field in the requested actions field. Each new activation of a DigitMap begins at step 1 of the above procedure, with a clear current dial string. Any previous contents of the current dial string from an earlier activation are lost.

A DigitMap completion event that does not contain an EventDM field in its requested actions field is considered an error. Upon receipt of such an event in an Events Descriptor, a MG shall respond with an error response, including Error Code 457 ("Missing parameter in signal or event").

7.1.14.7 Interaction of DigitMap and event processing

While the DigitMap is activated, detection is enabled for all events defined in the package containing the specified DigitMap completion event. Normal event behaviour (e.g., stopping of signals unless the digit completion event has the KeepActive flag enabled) continues to apply for each such event detected, except that:

- the events in the package containing the specified DigitMap completion event other than the completion event itself are not individually notified and have no side-effects unless separately enabled; and
- an event that triggers a partial match completion event is not recognized and therefore has no side effects until reprocessed following the recognition of the DigitMap completion event. Similarly buffered digit events are not recognized and have no side effects until processed.

7.1.14.8 Wildcards

Note that if a package contains a DigitMap completion event, then an event specification consisting of the PackageID with a wildcarded EventID will activate a DigitMap; to that end, the event specification must include an EventDM field according to 7.1.14.6. If the package also contains the digit events themselves, this form of event specification will cause the individual events to be reported to the MGC as they are detected.

7.1.14.9 Example

As an example, consider the following dial plan:

0	Local operator
00	Long-distance operator
xxxx	Local extension number (starts with 1-7)
8xxxxxxx	Local number
#xxxxxxx	Off-site extension
*xx	Star services
91xxxxxxxxxx	Long-distance number
9011 + up to 15 digits	International number

If the DTMF detection package described in E.6 is used to collect the dialled digits, then the dialling plan shown above results in the following DigitMap:

```
(0| 00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxxx|9011x.)
```

7.1.15 Statistics Descriptor

The Statistics Descriptor provides information describing the status and usage of a termination during its existence (ephemeral) or while it is outside the NULL context (physical). Statistics may be at a termination level or associated with a particular Stream Descriptor. By default, statistics occur at a termination level. If a stream is unable to support a statistic, Error Code 460 ("Unable to set statistic on stream") shall be returned. There is a set of standard statistics kept for each termination where appropriate (number of octets sent and received for example). By default, the particular statistics that are reported for a given termination are determined by the Packages realized by the termination. It is also possible, using the descriptor, to indicate what statistics are to be collected. The setting of a Statistics Descriptor overrides a previously set Statistics Descriptor. Thus, to maintain already set statistics, these shall be included in the new Statistics Descriptor and the statistic values shall not be reset. Statistics that have been removed from a Statistics Descriptor shall maintain their values until the termination is subtracted. However, if the particular statistic is reactivated by a subsequent Statistics Descriptor, the value shall be reset. A MGC may reactivate all statistics on a termination/stream by issuing a Statistics Descriptor with a single statistic with the PackageID and StatisticID wildcarded with "ALL". To reactivate all statistics in a given package on a termination/stream, a Statistics Descriptor with a single statistic with the PackageID specified and the StatisticID wildcarded with "ALL" is issued. The receipt of an empty descriptor means that no statistics shall be collected for the specified termination.

By default, unless an empty Statistics Descriptor had earlier been used to indicate that no statistics are to be collected, termination and stream level statistics are reported when the termination ceases to exist or is returned to the NULL context due to a Subtract command. This default behaviour can also be overridden by including an empty Audit Descriptor in the Subtract Command. If a stream is removed according to 7.1.6, by default stream level statistics are not reported. An audit of the Statistics Descriptor in the stream must be performed before the stream is removed to collect the statistics. Statistics may also be returned from the AuditValue Command, or any Add/Move/Modify Command using the Audit Descriptor.

Statistics are cumulative; reporting statistics does not reset them. The value of a Statistic at a termination level is the result of a meaningful superior function (like, for instance, sum or average) of the values as if it had been placed on all the streams in the termination. Such a superior function is dependent on the particular statistic type. Unless specified otherwise in the package that defines a particular statistic, the default behaviour is a sum of the values. Statistics are reset when a termination ceases to exist or is returned to the NULL context due to a Subtract Command.

7.1.16 Packages Descriptor

Used only with the AuditValue command, the Packages Descriptor returns a list of packages realized by the termination.

7.1.17 ObservedEvents Descriptor

ObservedEvents is supplied with the Notify Command to inform the MGC of which event(s) were detected. Used with the AuditValue Command, the ObservedEvents Descriptor returns events in the event buffer which have not been notified. ObservedEvents contains the RequestID of the Events Descriptor that triggered the notification, the event(s) detected, optionally the detection time(s) and any parameters of the observed event. Detection times are reported with a precision of hundredths of a second.

7.1.18 Topology Descriptor

A Topology Descriptor is used to specify flow directions between terminations in a context. Contrary to the descriptors in previous clauses, the Topology Descriptor applies to a context instead of a termination. The default topology of a context is that each termination's transmission is received by all other terminations. The Topology Descriptor is optional to implement. An MG that does not support Topology Descriptors, but receives a command containing one, returns Error Code 444 ("Unsupported or unknown descriptor"), and optionally includes a string containing the name of the unsupported descriptor ("Topology") in the error text in the Error Descriptor.

The Topology Descriptor occurs before the commands in an action. It is possible to have an action containing only a Topology Descriptor, provided that the context to which the action applies already exists.

A Topology Descriptor consists of a sequence of associated terminations of the form (*T1*, *T2*, *association[,StreamID]*). *T1* and *T2* specify terminations within the context, possibly using the ALL or CHOOSE wildcard. If the optional StreamID field is used, the association applies only to the particular stream between *T1* and *T2* labeled by the StreamID. If the StreamID field is omitted, the topology applies to all streams in the termination. The *association* specifies how media flows between these two terminations as follows:

- (*T1*, *T2*, isolate) means that the terminations matching *T2* do not receive media from the terminations matching *T1*, nor vice versa.
- (*T1*, *T2*, oneway) means that the terminations that match *T2* receive media from the terminations matching *T1*, but not vice versa. In this case, use of the ALL wildcard such that there are terminations that match either *T1* or *T2 but not both* is allowed.
- (*T1*, *T2*, onewayexternal) means the terminations that match *T2*, receive media sent externally by terminations matching *T1*, but not vice versa. In this case, use of the ALL wildcard for *T1* is not allowed.
- (*T1*, *T2*, onewayboth) means the terminations that match *T2*, receive media sent and received externally by terminations matching *T1*, but not vice versa. In this case, use of the ALL wildcard for *T1* and/or *T2* is not allowed.
- (*T1*, *T2*, bothway) means that the terminations matching *T2* receive media from the terminations matching *T1*, and vice versa. In this case it is allowed to use wildcards such that there are terminations that match both *T1* and *T2*. However, if there is a termination that matches both, no loopback is introduced.

CHOOSE wildcards may be used in *T1* and *T2* as well, under the following restrictions:

- the action (see clause 8) of which the Topology Descriptor is part contains an Add Command in which a CHOOSE wildcard is used;
- if a CHOOSE wildcard occurs in *T1* or *T2*, then a partial name shall not be specified.

The CHOOSE wildcard in a Topology Descriptor matches the TerminationID that the MG assigns in the first Add Command that uses a CHOOSE wildcard in the same action. An existing termination that matches *T1* or *T2* in the context to which a termination is added is connected to the newly added termination as specified by the Topology Descriptor. If a termination is not mentioned within a Topology Descriptor, any topology associated with it remains unchanged. If, however, a new termination is added into a context, its association with the other terminations within the context defaults to bothway, unless a Topology Descriptor is given to change this (e.g., if *T3* is added to a context with *T1* and *T2* with topology (*T3*, *T1*, oneway) it will be connected bothway to *T2*).

If the topology is applied to one particular stream (*T1*, *T2*, association, StreamID), the topology of other streams between the terminations does not change.

A Topology Descriptor shall not include a combination of associations between two terminations (Ti, Tj) with and without the optional StreamID field, to avoid undefined behaviour. For example (*T1*, *T2*, bothway) and (*T1*, *T2*, isolate, S1) shall not appear in the same descriptor. Upon receipt of such a Topology Descriptor, a MG shall respond with an error response, including Error Code 421 ("Unknown action or illegal combination of actions").

A oneway connection must be implemented in such a way that the other terminations in the context are not aware of the change in topology.

Figure 7, the table following it and Figure 8 following it show some examples of the effect of including Topology Descriptors in actions. In these examples it is assumed that the Topology Descriptors are applied in sequence. Figures 9 and 10 are stand-alone examples showing the specific effects of the onewayexternal and onewayboth topology settings.

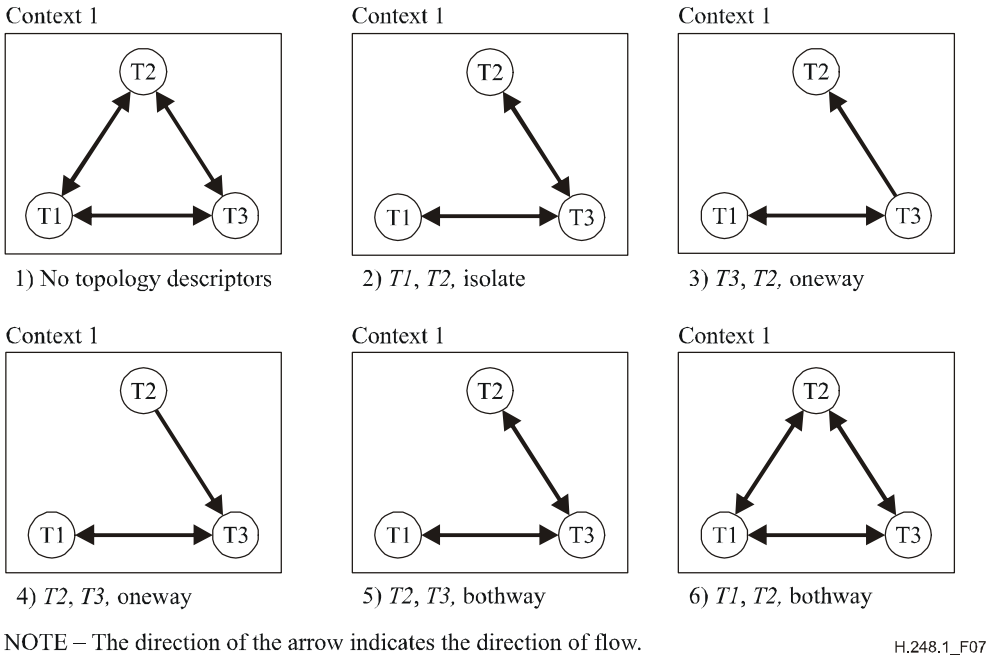

Figure 7/H.248.1 – Example topologies

Topology	Description
1	No Topology Descriptors When no Topology Descriptors are included, all terminations have a bothway connection to all other terminations.
2	*T1*, *T2* Isolate Removes the connection between *T1* and *T2*. *T3* has a bothway connection with both *T1* and *T2*. *T1* and *T2* have bothway connection to *T3*.
3	*T3*, *T2* oneway An oneway connection from *T3* to *T2* (i.e., *T2* receives media flow from *T3*). A bothway connection between *T1* and *T3*.
4	*T2*, *T3* oneway An oneway connection between *T2* to *T3*. *T1* and *T3* remain bothway connected
5	*T2*, *T3* bothway *T2* is bothway connected to *T3*. This results in the same as Topology 2.
6	*T1*, *T2* bothway (*T2*, *T3* bothway and *T1*, *T3* bothway may be implied or explicit). All terminations have a bothway connection to all other terminations.

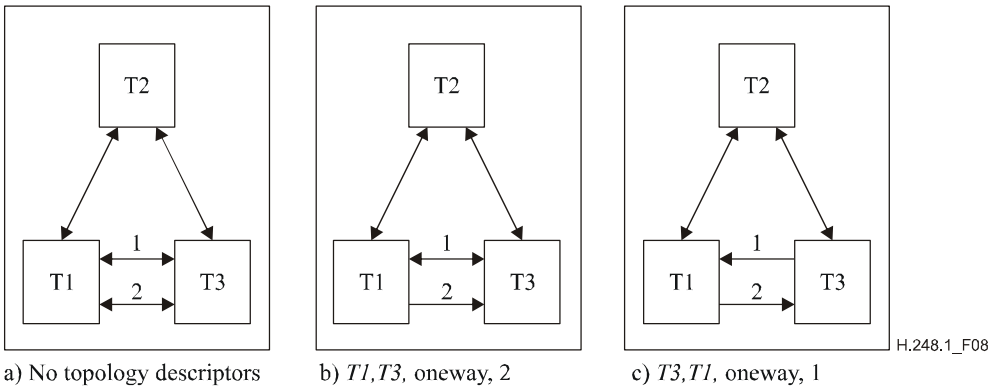

Figure 8/H.248.1 – Example topology at stream level

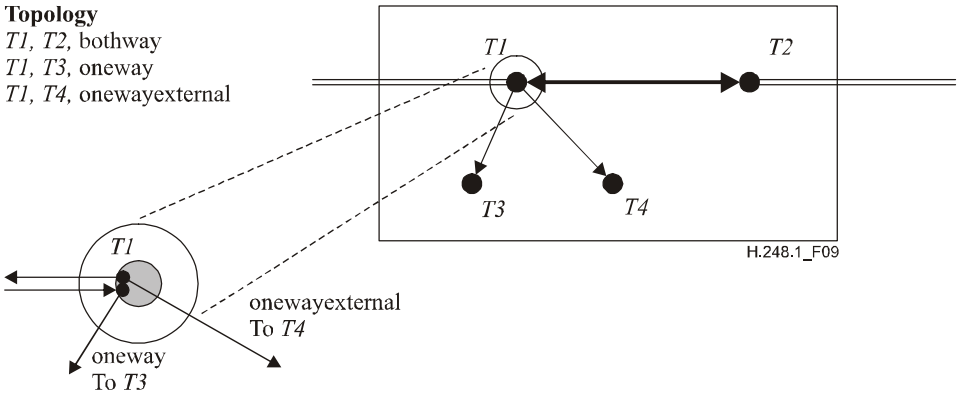

Figure 9/H.248.1 – Onewayexternal contrasted with oneway topology

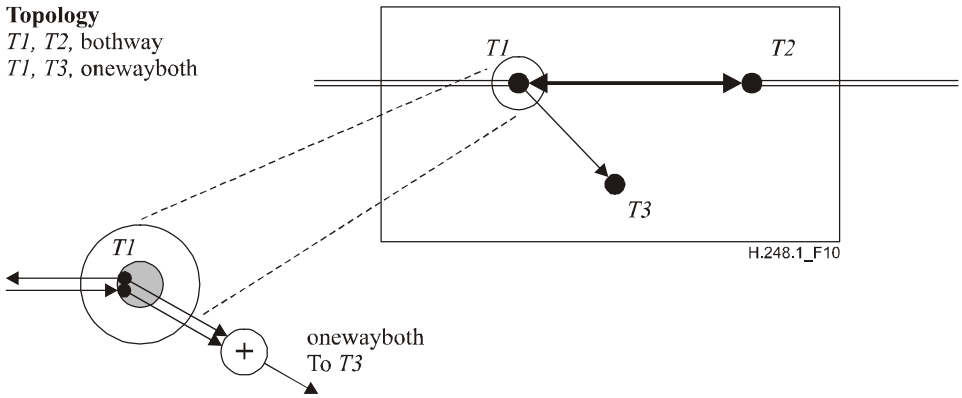

Figure 10/H.248.1 – Operation of onewayboth topology

7.1.19 ContextAttribute Descriptor

A ContextAttribute Descriptor is used to specify properties (defined in packages) that apply to the context as a whole and is applied on a context rather than a termination. Termination properties are not valid in the ContextAttribute Descriptor. The ContextAttribute Descriptor occurs before the commands in an action. It is possible to have an action containing only a ContextAttribute Descriptor, provided that the indicated context already exists. Values of context properties may be underspecified as in 7.1.1.

A new setting of the ContextAttribute Descriptor completely replaces the previous setting of that descriptor in the MG. Thus to retain information from the previous setting the MGC must include that information in the new setting. If the MGC wishes to delete some information from the existing descriptor, it merely resends the descriptor with the unwanted information stripped out. The inclusion of a ContextAudit or a ContextAttribute Descriptor in an action that contains only an AuditValue or AuditCapabilities Command does not constitute a new setting.

The ContextIDList parameter is used to provide a compact list of ContextIDs. To request the use of the ContextIDList parameter in a reply, the ContextIDList parameter should be included in the corresponding request.

7.1.20 Error Descriptor

If a responder encounters an error when processing a transaction request, it must include an Error Descriptor in its response. A Notify request may contain an Error Descriptor as well.

An Error Descriptor consists of an IANA-registered error code, optionally accompanied by an error text. ITU-T Rec. H.248.8 contains a list of valid error codes and error descriptions.

An Error Descriptor shall be specified at the "deepest level" that is semantically appropriate for the error being described and that is possible given any parsing problems with the original request. An Error Descriptor may refer to a syntactic construct other than where it appears. For example, Error Descriptor 422 ("Syntax Error in Action"), could appear within a command even though it refers to the larger construct, the action, and not the particular command within which it appears.

7.2 Command application programming interface

Following is an Application Programming Interface (API) describing the commands of the protocol. This API is shown to illustrate the commands and their parameters and is not intended to specify implementation (e.g., via use of blocking function calls). It describes the input parameters in parentheses after the command name and the return values in front of the command. This is only for descriptive purposes; the actual command syntax and encoding are specified in later subclauses. The order of parameters to commands is not fixed. Descriptors may appear as parameters to commands in any order. The descriptors shall be processed in the order in which they appear.

Any reply to a command may contain an Error Descriptor; the API does not specifically show this.

All parameters enclosed by square brackets ([. . .]) are considered optional.

7.2.1 Add

The Add Command adds a termination to a context.

TerminationIDList
[,MediaDescriptor]
[,ModemDescriptor] (*)
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]

```
Add( TerminationIDList
    [, MediaDescriptor]
    [, ModemDescriptor] (*)
    [, MuxDescriptor]
    [, EventsDescriptor]
    [, EventBufferDescriptor]
    [, SignalsDescriptor]
    [, DigitMapDescriptor]
    [, AuditDescriptor]
    [, StatisticsDescriptor]
)
```

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

The TerminationIDList specifies one or more terminations to be added to the context. The termination(s) is either created, or taken from the NULL Context. If a CHOOSE wildcard is used in the TerminationID, the selected TerminationID will be returned. ALL wildcards may be used in an Add, but such usage would be unusual. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified.

The optional Media Descriptor describes all media streams.

The optional Mux Descriptor specifies a multiplexer if applicable. For convenience, if a Multiplex Descriptor is present in an Add Command and lists any terminations that are not currently in the context, such terminations are added to the context as if individual Add Commands listing the terminations were invoked. If an error occurs on such an implied Add, error 471 ("Implied Add for Multiplex failure") shall be returned and further processing of the command shall cease.

The Events Descriptor parameter is optional. If present, it provides the list of events that should be detected on the termination.

The EventBuffer Descriptor parameter is optional. If present, it provides the list of events that the MG is requested to detect and buffer when EventBufferControl equals LockStep.

The Signals Descriptor parameter is optional. If present, it provides the list of signals that should be applied to the termination.

The DigitMap Descriptor parameter is optional. If present, it defines a DigitMap definition that may be used in an Events Descriptor.

The Audit Descriptor is optional. If present, the command will return the values of any descriptors/properties/signals/events/statistics specified in the Audit Descriptor.

The Statistics Descriptor is optional. If present, it provides the statistics which the MGC wishes to be collected for the termination or stream. Each time the statistic is set, the statistic value is reset.

All descriptors that can be modified could be returned by MG if a parameter was underspecified or overspecified. The ObservedEvents, Statistics, Packages and EventBuffer Descriptors are returned only if requested in the Audit Descriptor.

Add shall not be used on a termination with a serviceState of "OutOfService".

7.2.2 Modify

The Modify Command modifies the properties of a termination.

TerminationIDList

[,MediaDescriptor]

[,ModemDescriptor] (*)

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

```
Modify( TerminationIDList
    [, MediaDescriptor]
    [, ModemDescriptor] (*)
    [, MuxDescriptor]
    [, EventsDescriptor]
    [, EventBufferDescriptor]
    [, SignalsDescriptor]
    [, DigitMapDescriptor]
    [, AuditDescriptor]
    [, StatisticsDescriptor]
)
```

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

The TerminationIDList specifies the terminations to be modified. The TerminationID may be specific if a single termination in the context is to be modified. Use of wildcards in the

TerminationID may be appropriate for some operations. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. The CHOOSE TerminationID is an error, as the Modify command may only be used on existing terminations.

For convenience, if a Multiplex Descriptor is present in a Modify command, then:

- if the new Multiplex Descriptor lists any terminations that are not currently in the context, such terminations are added to the context as if individual Add Commands listing the terminations were invoked;
- if any terminations listed previously in the Multiplex Descriptor are no longer present in the new Multiplex Descriptor, they are subtracted from the context as if individual Subtract commands listing the terminations were invoked.

The remaining parameters to Modify are the same as those to Add. Possible return values are the same as those to Add.

7.2.3 Subtract

The Subtract Command disconnects a termination from its context and returns statistics on the termination's participation in the context.

TerminationIDList

[,MediaDescriptor]

[,ModemDescriptor] (*)

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

```
Subtract(TerminationIDList
    [, AuditDescriptor]
)
```

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

TerminationIDList in the input parameters represents the termination(s) that is being subtracted. The TerminationID may be specific or may be a wildcard value indicating that all (or a set of related) terminations in the context of the Subtract Command are to be subtracted. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified.

The use of CHOOSE in the TerminationID is an error, as the Subtract command may only be used on existing terminations.

ALL may be used as the ContextID as well as the TerminationID in a Subtract, which would have the effect of deleting all contexts, deleting all ephemeral terminations, and returning all physical terminations to NULL Context. Subtract of a termination from the NULL Context is not allowed.

For convenience, if a multiplexing termination is the object of a Subtract command, then any bearer terminations listed in its Multiplex Descriptor are subtracted from the context as if individual Subtract commands listing the terminations were invoked.

By default unless overridden, the Statistics Descriptor on both the termination and stream levels is returned to report information collected on the termination or terminations specified in the command. The information reported applies to the termination's or terminations' existence in the context from which it or they are being subtracted.

The Audit Descriptor is optional. If present, the command will return only those items specified in the Audit Descriptor, which may be empty. If the Audit Descriptor is omitted, the Statistics Descriptor is returned, by default. Possible return values are the same as those to Add.

When a provisioned termination is Subtracted from a context, its descriptor values shall revert to:

- the default value, if specified for the descriptor and not overridden by provisioning;
- otherwise, the provisioned value.

7.2.4 Move

The Move Command moves a termination to another context from its current context in one atomic operation. The Move Command is the only command that refers to a termination in a context different from that to which the command is applied. The Move Command shall not be used to move terminations to or from the NULL Context.

TerminationIDList

[,MediaDescriptor]

[,ModemDescriptor] (*)

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

```
Move( TerminationIDList
    [, MediaDescriptor]
    [, ModemDescriptor] (*)
    [, MuxDescriptor]
    [, EventsDescriptor]
    [, EventBufferDescriptor]
    [, SignalsDescriptor]
    [, DigitMapDescriptor]
    [, AuditDescriptor]
    [, StatisticsDescriptor]
)
```

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

The TerminationIDList specifies the termination(s) to be moved. TerminationIDs may be wildcarded, but CHOOSE shall not be used. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. The context to which the termination is moved is

indicated by the target ContextID in the action. If the last remaining termination is moved out of a context, the context is deleted.

The Move Command does not affect the properties of the termination on which it operates, except those properties explicitly modified by descriptors included in the Move Command. The Audit Descriptor with a Statistics Descriptor, for example, would return statistics on the termination just prior to the Move. Possible descriptors returned from Move are the same as for Add.

For convenience, if a multiplexing termination is the object of a Move Command, then any bearer terminations listed in its Multiplex Descriptor are also moved as if individual Move Commands listing the terminations were invoked.

Move shall not be used on a termination with a serviceState of "OutOfService".

7.2.5 AuditValue

The AuditValue Command returns the current values of properties, events, signals and statistics associated with terminations. An AuditValue may request the contents of a descriptor or of a single property, event, signal or statistic. AuditValue may be useful for maintaining termination synchronization between the MGC and MG. An AuditValue may request that the returned values be filtered based on specified selection criteria.

TerminationIDList

[,MediaDescriptor]

[,ModemDescriptor] (*)

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

```
AuditValue(TerminationIDList,
    AuditDescriptor
)
```

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

TerminationIDList specifies a sequence of TerminationIDs, which may be specific or wildcarded. If the wildcard matches more than one TerminationID, all possible matches are attempted, with results reported for each one. The order of attempts when multiple TerminationIDs match is not specified. If a wildcarded response is requested, only one command return is generated, with the contents containing the union of the values of all terminations matching the wildcard or list. This convention may reduce the volume of data required to audit a group of terminations. Use of CHOOSE is an error.

Descriptors or individual properties, signals, events and statistics can be audited. The returned values can be filtered based on specified selection criteria.

- An audit of a descriptor may be requested by identifying the desired descriptor in the Audit Descriptor with no further information.
- To audit an individual property in the Media Descriptor the relevant StreamID (optional), GroupID (optional) and PropertyID are included. The current value of the property is

returned. GroupID is used in the case where the LocalControl Descriptor ReserveGroup flag is used. GroupID 1 corresponds to the first group (session description) reserved, GroupID 2 the next group, etc.

- To audit a signal, the relevant SignalListID and/or SignalID are provided. The StreamID and/or RequestID are optional. Only if the signal is active are the values of all the signal parameters returned including: the KeepActive indication, signal type, duration, signal completion indication and package defined properties.
- To audit an event, the relevant StreamID (optional), EventID, RequestID (optional) are provided. The values of all the event parameters are returned including: event actions and packaged defined parameters.
- To audit a termination level statistic, the identity of the statistic is provided. To audit a stream level statistic, the relevant StreamID (optional), GroupID (optional) and the StatisticID is included in the Media Descriptor. The current value of the statistic is returned. The statistic is not reset.
- To audit a package, the identity and version of the package is provided. All properties, signals, events and statistics defined in that particular package are returned with their current value.
- To filter the audit return values, selection criteria are included in the audit request. It is possible to include multiple selection criteria. When multiple criteria are included, either an AND or an OR logic operation may also be included to indicate how the selection criteria are to be interpreted. If no logic operation is included, an AND logic operation is assumed.

It is possible to audit multiple individual items in one request.

If a descriptor audit is requested, the appropriate descriptors, with the current values for the termination, are returned from AuditValue. Values appearing in multiple instances of a descriptor are defined to be alternate values supported, with each parameter in a descriptor considered independent.

ObservedEvents returns a list of events in the EventBuffer. If the ObservedEvents Descriptor is audited while a DigitMap is active, the returned ObservedEvents Descriptor also includes a DigitMap completion event that shows the current dial string but does not show a termination method.

EventBuffer returns the set of events and associated parameter values currently enabled in the EventBuffer Descriptor. Packages Descriptor returns a list of packages realized by the termination. DigitMap Descriptor returns the name or value of the current DigitMap for the termination. DigitMap requested in an AuditValue command with TerminationID ALL returns all DigitMaps in the gateway. Statistics returns the current values of all statistics being kept at the termination level. An audit of the Media Descriptor will return any stream level statistics. Specifying an empty Audit Descriptor results in only the TerminationID being returned. This may be useful to get a list of TerminationIDs when used with wildcarding. Annexes A and B provide a special syntax for presenting such a list in condensed form, such that the AuditValue command tag does not have to be repeated for each TerminationID.

AuditValue results depend on the context, viz. specific, NULL, or wildcarded. (Note that ContextID ALL does not include the NULL Context.) The TerminationID may be specific, or wildcarded.

The following are examples of what is returned in case the context and/or the termination is wildcarded and a wildcarded response has been specified.

Assume that the gateway has 4 terminations: t1/1, t1/2, t2/1 and t2/2. Assume that Terminations t1/* have implemented Packages aaa and bbb and that Terminations t2/* have implemented Packages ccc and ddd. Assume that Context 1 has t1/1 and t2/1 in it and that Context 2 has t1/2 and t2/2 in it.

The command:

Context=1{AuditValue=t1/1{Audit{Packages}}}

returns:

Context=1{AuditValue=t1/1{Packages{aaa,bbb}}}

The command:

Context=*{AuditValue=t2/*{Audit{Packages}}}

returns:

Context=1{AuditValue=t2/1{Packages{ccc,ddd}}},

Context=2{AuditValue=t2/2{Packages{ccc,ddd}}}

The command:

Context=*{W-AuditValue=t1/*{Audit{Packages}}}

returns:

Context=*{AuditValue=t1/*{Packages{aaa,bbb}}}

NOTE – A wildcard response may also be used for other commands such as Subtract.

In the instance of a wildcarded context with Root, the MGC can indicate that it wants a compact list of ContextIDs, rather than having each context broken out into separate action replies.

Assume that the gateway has 4 contexts: 1, 2, 3, 4, each containing two terminations (t1-t8). The following examples indicate how the MG would respond to the audit command:

The command:

Context=*{AuditValue=Root{Audit{}}}

returns:

Context=1{AuditValue=Context{*}}, Context=2{AuditValue=Context{*}},
Context=3{AuditValue=Context{*}}, Context=4{AuditValue=Context{*}}

Or

Context=1{AuditValue=t1{},AuditValue=t2{}},Context=2{AuditValue=t3{},AuditValue=t4{}},Context=3{AuditValue=t5{},AuditValue=t6{}},Context=4{AuditValue=t7{},AuditValue=t8{}}

The command:

Context=*{ContextAttr{ContextList={*}},AuditValue=Root{Audit{}}}

returns:

Context=*{ContextAttr{ContextList={1,2,3,4}},AuditValue=Root{}}

The following illustrates other information that can be obtained with the AuditValue Command:

ContextID	TerminationID	Information obtained
Specific	Wildcard	Audit of matching terminations in a context
Specific	Specific	Audit of a single termination in a context
NULL	Root	Audit of Media Gateway state and events
NULL	Wildcard	Audit of all matching terminations in the NULL Context
NULL	Specific	Audit of a single termination outside of any context
ALL	Wildcard	Audit of all matching terminations not in NULL and the context to which they are associated
ALL	Root	List of all ContextIDs (the ContextID list may be returned either by using multiple action replies, each containing a ContextID from the list, or using the ContextIDList parameter. The method of response is determined by the presence of the ContextIDList parameter in the request.)
ALL	Specific	(Non-NULL) ContextID in which the termination currently exists

7.2.6 AuditCapabilities

The AuditCapabilities Command returns the possible values of properties, events, signals and statistics associated with terminations. An AuditCapabilities may be requested for the contents of a descriptor or for a single property, event, signal or statistic.

TerminationIDList

[,MediaDescriptor]

[,ModemDescriptor](*)

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

AuditCapabilities(TerminationIDList,

AuditDescriptor)

(*) Modem Descriptor has been deprecated in H.248.1 version 2 (05/2002).

Descriptors or individual properties, signals, events and statistics can be audited.

- An audit of an entire descriptor may be requested by identifying the desired descriptor in the Audit Descriptor with no further information.
- To audit an individual property in the Media Descriptor the relevant StreamID (optional) and PropertyID are included. A list of possible values of the property are returned.
- To audit a signal the relevant SignalListID and/or SignalID are provided. The StreamID and/or RequestID are optional. A list of possible values associated with each signal parameter is returned (including package defined properties). The KeepActive indication, signal type, duration and signal completion indication are not returned.
- To audit an event the relevant StreamID (optional), EventID, RequestID (optional) are provided. A list of possible values associated with each event parameter is returned (including event actions and package defined parameters).

- To audit a termination level statistic the identity of the statistic is provided. To audit a stream level statistic, the relevant StreamID (optional), GroupID (optional) and the StatisticID is included in the Media Descriptor. The possible values of the statistic are returned. The statistic is not reset.

If a descriptor audit is requested, the appropriate descriptors, with the possible values for the termination, are returned from AuditCapabilities. Descriptors may be repeated where there are multiple possible values.

If a wildcarded response is requested, only one command return is generated, with the contents containing the union of the values of all terminations in the list or matching the wildcard. This convention may reduce the volume of data required to audit a group of terminations.

If a property, signal, event or statistic is audited, the appropriate properties, signals, events and statistics with the capabilities of the termination, are returned from AuditCapabilities.

Interpretation of what capabilities are requested for various values of ContextID and TerminationID is the same as in AuditValue.

For property and parameter values of type string, character or octet string, the MG shall return an empty value. For the text encoding, strings and characters return an empty quotedString construct, while octet strings return NULL (0x00). This behaviour may be overridden by the package definition.

The Events Descriptor returns the list of possible events on the termination together with the list of all possible values for the Events Descriptor Parameters. EventBuffer Descriptor returns the same information as Events Descriptor. The Signals Descriptor returns the list of possible signals that could be applied to the termination, together with the list of all possible values for the signals' parameters. Statistics Descriptor returns the names of the statistics being kept on the termination. ObservedEvents Descriptor returns the names of active events on the termination. DigitMap and Packages are not legal in AuditCapability.

The following illustrates other information that can be obtained with the AuditCapabilities Command:

ContextID	TerminationID	Information obtained
Specific	Wildcard	Audit of matching terminations in a context
Specific	Specific	Audit of a single termination in a context
NULL	Root	Audit of MG state and events
NULL	Wildcard	Audit of all matching terminations in the NULL Context
NULL	Specific	Audit of a single termination outside of any context
ALL	Wildcard	Audit of all matching terminations not in NULL and the context to which they are associated
ALL	Root	Same as for AuditValue
ALL	Specific	Same as for AuditValue

7.2.7 Notify

The Notify Command allows the Media Gateway to notify the Media Gateway Controller of events occurring within the Media Gateway.

TerminationID

Notify(TerminationID,

ObservedEventsDescriptor,

[ErrorDescriptor])

The TerminationID specifies the termination issuing the Notify Command. The TerminationID shall be a fully qualified name.

The ObservedEvents Descriptor contains the RequestID and a list of events that the Media Gateway detected in the order that they were detected. Each event in the list is accompanied by parameters associated with the event and, optionally, an indication of the time that the event was detected. Procedures for sending Notify commands with RequestID equal to zero are for further study.

Notify Commands with RequestID not equal to zero shall occur only as the result of detection of an event specified by an Events Descriptor which is active on the termination concerned.

The RequestID returns the RequestID parameter of the Events Descriptor that triggered the Notify Command. It is used to correlate the notification with the request that triggered it. The events in the list must have been requested via the triggering Events Descriptor or embedded Events Descriptor unless the RequestID is zero (which is for further study).

The Error Descriptor may be sent in the Notify Command as a result of Error 518 ("Event buffer full").

7.2.8 ServiceChange

The ServiceChange Command allows the Media Gateway to notify the Media Gateway Controller that a termination, or group of terminations, is about to be taken out of service or has just been returned to service. The Media Gateway Controller may indicate that termination(s) shall be taken out of or returned to service. The Media Gateway may notify the MGC that the capability of a termination has changed. It also allows a MGC to hand over control of a MG to another MGC.

TerminationIDList,

[ServiceChangeDescriptor]

ServiceChange(TerminationIDList,

ServiceChangeDescriptor

)

The TerminationIDList specifies the termination(s) that are taken out of or returned to service. Wildcarding of termination names is permitted, with the exception that the CHOOSE mechanism shall not be used. Use of the "Root" TerminationID indicates a ServiceChange affecting the entire Media Gateway.

NOTE – The use of TerminationIDList is not valid in the initial H.248.1 Version 1 ServiceChange Command.

7.2.8.1 ServiceChange Descriptor contents

The ServiceChange Descriptor contains the following parameters as required:

- ServiceChangeMethod;
- ServiceChangeReason;
- ServiceChangeDelay;

- ServiceChangeAddress;
- ServiceChangeProfile;
- ServiceChangeVersion;
- ServiceChangeMgcID;
- TimeStamp;
- ExtensionParameter;
- ServiceChangeInfo;
- ServiceChangeIncompleteFlag.

7.2.8.1.1 ServiceChangeMethod

The ServiceChangeMethod parameter specifies the type of ServiceChange that will or has occurred:

1) Graceful: indicates that the specified terminations will be taken out of service after the specified ServiceChangeDelay; established connections are not yet affected, but the Media Gateway Controller should refrain from establishing new connections and should attempt to gracefully tear down existing connections on the termination(s) affected by the ServiceChange Command. The MG should set the termination's serviceState at the expiry of ServiceChangeDelay or the removal of the termination from an active context (whichever is first), to "out of service".
2) Forced: indicates that the specified terminations were taken abruptly out of service and any established connections associated with them may be lost. For non-Root terminations, the MGC is responsible for cleaning up the context (if any) with which the failed termination is associated. At a minimum, the termination shall be subtracted from the context. The termination's serviceState should be "out of service". For the Root Termination, the MGC can assume that all connections are lost on the MG and thus can consider that all the terminations have been subtracted.
3) Restart: indicates that service will be restored on the specified terminations after expiration of the ServiceChangeDelay. The ServiceState should be set to "inService" upon expiry of ServiceChangeDelay.
4) Disconnected: always applied with the Root TerminationID, indicates that the MG lost communication with the MGC, but it was subsequently restored to the same MGC (possibly after trying other MGCs on a pre-provisioned list). Since MG state may have changed, the MGC may wish to use the Audit command to resynchronize its state with the MGs.
5) Handoff: sent from the MGC to the MG, this reason indicates that the MGC is going out of service and a new MGC association must be established. Sent from the MG to the MGC, this indicates that the MG is attempting to establish a new association in accordance with a Handoff received from the MGC with which it was previously associated.
6) Failover: sent from MG to MGC to indicate the primary MG is out of service and a secondary MG is taking over. This ServiceChange method is also sent from the MG to the MGC when the MG detects that MGC has failed.
7) Another value whose meaning is mutually understood between the MG and the MGC.

7.2.8.1.2 ServiceChangeReason

The ServiceChangeReason parameter specifies the reason why the ServiceChange has or will occur. It consists of an alphanumeric token (IANA registered) and, optionally, an explanatory string.

7.2.8.1.3 ServiceChangeAddress and ServiceChangeMgcID

The optional ServiceChangeAddress parameter specifies the address (e.g., IP port number for IP networks) to be used for subsequent communications. It can be specified in the input parameter

descriptor or the returned result descriptor. ServiceChangeAddress and ServiceChangeMgcID parameters must not both be present in the ServiceChange Descriptor or the ServiceChangeResult Descriptor. The ServiceChangeAddress provides an address to be used within the context of the association currently being negotiated, while the ServiceChangeMgcID provides an alternate address where the MG should seek to establish another association. Note that the use of ServiceChangeAddress is not encouraged. MGCs and MGs must be able to cope with the ServiceChangeAddress being either a full address or just a port number in the case of TCP transports.

7.2.8.1.4 ServiceChangeDelay

The optional ServiceChangeDelay parameter is expressed in seconds. If the delay is absent or set to zero, the delay value should be considered to be null. In the case of a "Graceful" ServiceChangeMethod, a null delay indicates that the Media Gateway Controller should wait for the natural removal of existing connections and should not establish new connections. For "Graceful" only, a null delay means the MG must not set ServiceState "OutOfService" until the termination is in the NULL Context.

7.2.8.1.5 ServiceChangeProfile

The optional ServiceChangeProfile parameter specifies the Profile (if any) of the protocol supported. The ServiceChangeProfile includes the version of the profile supported. In the absence of this parameter, the value "NoProfile" is assumed.

7.2.8.1.6 ServiceChangeVersion

The optional ServiceChangeVersion parameter contains the protocol version and is used if protocol version negotiation occurs (see 11.3).

7.2.8.1.7 TimeStamp

The optional TimeStamp parameter specifies the actual time as kept by the sender. As such, it is not necessarily absolute time according to, for example, a local time zone: it merely establishes an arbitrary starting time against which all future timestamps transmitted by a sender during this association shall be compared. It can be used by the responder to determine how its notion of time differs from that of its correspondent. TimeStamp is sent with a precision of hundredths of a second.

7.2.8.1.8 ExtensionParameter

The optional Extension parameter may contain any value whose meaning is mutually understood by the MG and MGC. The value "X-SC" is reserved for the use of signalling ServiceChange parameters that have been added as a result of H.248.1 version 3 (and subsequent versions). It is only used in the initial H.248.1 version 1 coded ServiceChange Command from the MG to MGC when the ServiceChangeVersion is greater than or equal to 3. The structure of the value is defined by the following ABNF:

X-SC = 1* (NAME EQUAL paramValue [COMMA])

When adding new ServiceChange parameters it is recommended that a name/value be assigned if the parameter is to be sent in an initial ServiceChange. For example:

Name: Strings of up to 64 characters, containing no spaces, starting with an alphabetic character and consisting of alphanumeric characters and/or digits, and possibly including the special character underscore ("_").

Type: As per 12.1.2 "Properties"

Possible values: As per 12.1.2 "Properties"

7.2.8.1.9 ServiceChangeInfo

The optional ServiceChangeInfo parameter may contain the package/property/signal/event/statistic of the reason that caused the ServiceChange.

7.2.8.1.10 ServiceChangeIncompleteFlag

This optional flag indicates that subsequent ServiceChange Commands will be sent from the MG to the MGC indicating the state of terminations. It is only used during MG registration or restart (ServiceChange on Root with ServiceChangeMethod Restart) when the MG wants to report the entire MG and termination state to the MGC. Upon reception of this flag, the MGC shall refrain from generating commands operating on terminations other than Root to the MG. After sending the ServiceChangeIncompleteFlag in the initial registration/restart command, the MG shall send it in all subsequent ServiceChange Commands until the MG has determined that it has reported the current state of the MG and its terminations. The flag shall then be removed from the last ServiceChange Command indicating the state of the MG or its terminations. Upon the reception of this ServiceChange without the ServiceChangeIncompleteFlag, the MGC may again generate commands to the MG. When this flag is sent in an initial ServiceChange (H.248.1 Version 1) Command using the ServiceChange extension parameter "X-SC" the following is used:

Name: SIC

Type: Boolean

Possible values: ON: Flag included

NOTE – The value OFF is not required as subsequent commands will be H.248.1 Version 3 encoded.

7.2.8.1.11 ServiceChange Command and Response

A ServiceChange Command specifying the "Root" for the TerminationID and ServiceChangeMethod equal to Restart is a registration command by which a Media Gateway announces its existence to the Media Gateway Controller. The Media Gateway may register by specifying the Root TerminationID and ServiceChangeMethod equal to Failover when the MG detects MGC failures. A message containing a ServiceChange Command specifying the Root TerminationID and ServiceChangeMethod equal to Restart or Failover shall not contain other commands as these commands should use the new ServiceChangeAddress and negotiated protocol version.

The Media Gateway is expected to be provisioned with the name of one primary and optionally some number of alternate Media Gateway Controllers.

Acknowledgement of the ServiceChange Command completes the registration process, except when the MGC has returned an alternative ServiceChangeMgcID as described below.

The MG may specify the transport ServiceChangeAddress to be used by the MGC for sending messages in the ServiceChangeAddress parameter in the input ServiceChange Descriptor. The MG may specify an address in the ServiceChangeAddress parameter of the ServiceChange request, and the MGC may also do so in the ServiceChange reply. In either case, the recipient must use the supplied address as the destination for all subsequent transaction requests within the association. At the same time, as indicated in clause 9, transaction replies and pending indications must be sent to the address from which the corresponding requests originated. This must be done even if it implies extra messaging because commands and responses cannot be packed together.

The TimeStamp parameter shall be sent with a registration command and its response.

At registration the MG may send the ServiceChangeIncompleteFlag if it has determined that the MG is in state "InService" but individual terminations may be in state "OutOfService". This is to prevent attempted seizure of out-of-service terminations. Subsequent ServiceChange Commands are used to report the "OutOfService" terminations.

The Media Gateway Controller may return a ServiceChangeMgcID parameter that describes the Media Gateway Controller that should preferably be contacted for further service by the Media Gateway. In this case, the Media Gateway shall reissue the ServiceChange Command to the new Media Gateway Controller. The MGC specified in a ServiceChangeMgcID, if provided, shall be contacted before any further alternate MGCs. On a handoff message from MGC to MG, the ServiceChangeMgcID is the new MGC that will take over from the current MGC.

The return from ServiceChange is empty except when the Root TerminationID is used. In that case, it includes the following parameters as required:

- ServiceChangeAddress, if the responding MGC wishes to specify a new destination for messages from the MG for the remainder of the association;
- ServiceChangeMgcID, if the responding MGC does not wish to sustain an association with the MG;
- ServiceChangeProfile, if the responder wishes to negotiate the profile to be used for the association. The profile (name and version) is only returned in the reply in the case that the MGC cannot support the specified profiles in the ServiceChangeRequest. The returned reply shall indicate the profile and version supported or "NoProfile" if no profile is supported. Upon reception of a profile in the reply the MG may continue the relationship with the current MGC or contact secondary MGCs and establish a relationship with them. If the MGC returned a profile in the reply other than the one that was provided in the request the MG shall:
  a) continue the control association by issuing a new ServiceChange Command with an agreed profile to confirm to the MGC that the MG has agreed with the profile; or
  b) keep the control association active, so that the MGC will use the profile that it sent in the ServiceChange Reply; or
  c) initiate a control association with a different MGC using its original profile.
- ServiceChangeVersion, if the responder wishes to negotiate the version of the protocol to be used for the association.

ServiceChangeReasons are defined in ITU-T Rec. H.248.8. The list may be extended by an IANA registration as outlined in 14.3.

7.2.9 Manipulating and auditing context attributes

The commands of the protocol as discussed in the preceding subclauses apply to terminations. This clause specifies how contexts are manipulated and audited.

An action may contain context manipulation and auditing instructions (see clause 8).

An action request sent to a MG may include a request to audit attributes of a context. There are two types of audit:

Value Audit: The MGC may audit a specific context to determine the current value of individual context properties. The MGC may determine the current values for all existing (non-NULL) contexts by specifying ContextID ALL in the Audit request. If context attributes are added, or have been modified by the same action as the Audit request, the value(s) returned shall be after the action has been applied. Context attributes can be used as selection criteria to filter the audit return values. It is possible to include multiple selection criteria. When multiple criteria are included, either an AND or an OR logic operation may also be included to indicate how the selection criteria are to be interpreted. If no logic operation is included, an AND logic operation is assumed.

Capabilities Audit: The MGC may audit a specific context to determine the possible values that individual context properties can assume. The MGC may determine the possible values for the entire MG by specifying ContextID ALL in the audit request.

The following illustrates information that can be obtained with a Context Audit:

ContextID	TerminationID	AuditValue	AuditCapabilities
Specific	Not Applicable	Context property's value in the specified context.	Context property's possible values in the specified context.
NULL	Not Applicable	Not Allowed	Not Allowed
ALL	Not Applicable	Current values for all existing (non-NULL) contexts by specifying ContextID ALL in the Audit request. A response for ContextID ALL is presented through an actionReply per context.	A response of ContextID ALL is returned with context property's possible values across the entire MG.

An action may also include a request to change the attributes of a context.

The context properties that may be included in an action reply are used to return information to a MGC. This can be information requested by an audit of context attributes or details of the effect of manipulation of a context.

If a MG receives an action that contains both a request to audit context attributes and a request to manipulate those attributes, the response shall include the values of the attributes after processing the manipulation request.

7.2.10 Generic command syntax

The protocol can be encoded in a binary format or in a text format. MGCs should support both encoding formats. MGs may support both formats.

The protocol syntax for the binary format of the protocol is defined in Annex A. Annex C specifies the encoding of the Local and Remote Descriptors for use with the binary format.

A complete ABNF of the text encoding of the protocol per RFC 2234 is given in Annex B. SDP is used as the encoding of the Local and Remote Descriptors for use with the text encoding as modified in 7.1.8.

8 Transactions

Commands between the Media Gateway Controller and the Media Gateway are grouped into transactions, each of which is identified by a TransactionID. Transactions consist of one or more actions. An action consists of a non-empty series of commands, context property modifications, or context property audits that are limited to operating within a single context. Consequently, each action typically specifies a ContextID. However, there are two circumstances where a specific ContextID is not provided with an action. One is the case of modification of a termination outside of a context. The other is where the controller requests the gateway to create a new context. Figure 11 is a graphic representation of the transaction, action and command relationships.

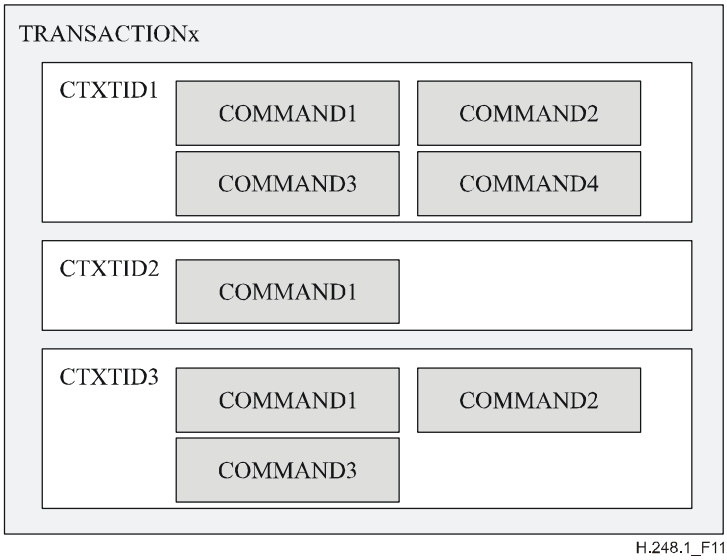

Figure 11/H.248.1 – Transactions, actions and commands

Transactions are presented as TransactionRequests. Corresponding responses to a TransactionRequest are received in a single reply, possibly preceded by a number of TransactionPending messages (see 8.2.3).

Transactions guarantee ordered command processing. That is, commands within a transaction are executed sequentially. Ordering of transactions is not guaranteed; transactions may be executed in any order, or simultaneously; however, TransactionReplies should be executed before TransactionRequests when both are contained in a message.

At the first failing command in a transaction, processing of the remaining commands in that transaction stops. If a command contains a wildcarded TerminationID, the command is attempted with each of the actual TerminationIDs matching the wildcard. A response within the TransactionReply is included for each matching TerminationID, even if one or more instances generated an error. If any TerminationID matching a wildcard results in an error when executed, any commands following the wildcarded command are not attempted.

Commands may be marked as "Optional" which can override this behaviour – if a command marked as Optional results in an error, subsequent commands in the transaction will be executed. If a command fails, the MG shall, as far as possible, restore the state that existed prior to the attempted execution of the command before continuing with command processing.

A TransactionReply includes the results for all of the commands in the corresponding TransactionRequest. The TransactionReply includes the return values for the commands that were executed successfully, and the command and Error Descriptor for any command that failed. Command replies are returned in the order in which they appeared in the corresponding TransactionRequest. TransactionPending is used to periodically notify the receiver that a transaction has not completed yet, but is actively being processed.

Applications should implement an application level timer per transaction. Expiration of the timer should cause a retransmission of the request. Receipt of a Reply should cancel the timer. Receipt of Pending should restart the timer.

8.1 Common parameters

8.1.1 Transaction identifiers

Transactions are identified by a TransactionID, which is assigned by sender and is unique within the scope of the sender. A response containing an Error Descriptor to indicate that the TransactionID is missing in a request shall use a TransactionID of zero in the corresponding TransactionReply.

8.1.2 Context identifiers

Contexts are identified by a ContextID, which is assigned by the Media Gateway and is unique within the scope of the Media Gateway. The Media Gateway Controller shall use the ContextID supplied by the Media Gateway in all subsequent transactions relating to that context. The protocol makes reference to a distinguished value that may be used by the Media Gateway Controller when referring to a termination that is currently not associated with a context, namely the *NULL* ContextID.

The CHOOSE wildcard is used to request that the Media Gateway create a new context.

The MGC may use the ALL wildcard to address all contexts on the MG. The NULL Context is not included when the ALL wildcard is used.

The MGC shall not use partially specified ContextIDs containing the CHOOSE or ALL wildcards.

8.2 Transaction application programming interface

Following is an Application Programming Interface (API) describing the transactions of the protocol. This API is shown to illustrate the transactions and their parameters and is not intended to specify implementation (e.g., via use of blocking function calls). It will describe the input parameters and return values expected to be used by the various transactions of the protocol from a very high level. Transaction syntax and encodings are specified in later subclauses.

8.2.1 TransactionRequest

The TransactionRequest is invoked by the sender. There is one transaction per request invocation. A request contains one or more actions, each of which specifies its target context.

TransactionRequest(TransactionID {

ContextID {Command … Command},

. . .

ContextID {Command … Command}})

The TransactionID parameter must specify a value for later correlation with the TransactionReply or TransactionPending response from the receiver.

The ContextID parameter must specify a value to pertain to all commands that follow up to either the next specification of a ContextID parameter or the end of the TransactionRequest, whichever comes first.

The command parameter represents one of the commands mentioned in clause 7.2 ("Command application programming interface").

8.2.2 TransactionReply

The TransactionReply is invoked by the receiver. There is one reply invocation per transaction. A reply contains one or more actions, each of which must specify its target context and one or more Responses per context. A reply may be segmented into multiple messages. The TransactionReply is invoked by the responder when it has processed the TransactionRequest.

A TransactionRequest has been processed:

- when all actions in that TransactionRequest have been processed; or
- when an error is encountered in processing that TransactionRequest, except when the error is in an optional command.

If a message contains more TransactionRequests than the receiver can process, the receiver will return Error Descriptor 413 ("Number of transactions in message exceeds maximum").

A command has been processed when all descriptors in that command have been processed.

A Signals Descriptor is considered to have been processed when it has been established that the descriptor is syntactically valid, the requested signals are supported and they have been queued to be applied.

An Events Descriptor or EventBuffer Descriptor is considered to have been processed when it has been established that the descriptor is syntactically valid, the requested events can be observed, any embedded signals can be generated, any embedded events can be detected, and the MG has been brought into a state in which the events will be detected.

TransactionReply(TransactionID {

ContextID {Response … Response},

. . .

ContextID {Response … Response}})

The TransactionID parameter must be the same as that of the corresponding TransactionRequest.

The ContextID parameter must specify a value to pertain to all Responses for the action. The ContextID may be specific, ALL or NULL.

Each of the Response parameters represents a return value as mentioned in 7.2 or an Error Descriptor if the command execution encountered an error. Commands after the point of failure are not processed and, therefore, Responses are not issued for them.

An exception to this occurs if a command has been marked as optional in the TransactionRequest. If the optional command generates an error, the transaction still continues to execute, so the Reply would, in this case, have Responses after an Error.

Clause 7.1.20 ("Error Descriptor") specifies the generation of Error Descriptors. The text below discusses several individual cases.

If the receiver encounters an error in processing a ContextID but can parse the ContextID, the requested action response will consist of the ContextID and a single Error Descriptor, 422 ("Syntax Error in Action"). If the receiver cannot parse the ContextID, it will return a TransactionReply consisting of the TransactionID and a single Error Descriptor, 422 (Syntax Error in Action).

If the receiver encounters an error such that it cannot determine a legal action, it will return a TransactionReply consisting of the TransactionID and a single Error Descriptor, 422 ("Syntax Error in Action"). If the end of an action cannot be reliably determined but one or more commands can be parsed, the receiver will process the commands and then send 422 ("Syntax Error in Action") as the last action for the transaction. If the receiver encounters an error such that is cannot determine a legal transaction, it will return a TransactionReply with a NULL TransactionID and a single Error Descriptor 403 ("Syntax Error in TransactionRequest").

If the end of a transaction cannot be reliably determined and one or more actions can be parsed, the receiver will process the actions and then return 403 ("Syntax Error in TransactionRequest") as the last action reply for the transaction. If no actions can be parsed, it will return 403 ("Syntax Error in TransactionRequest") as the only reply.

If the TerminationID cannot be reliably determined, it will send 442 ("Syntax Error in Command") as the action reply.

If the end of a command cannot be reliably determined, it will return 442 ("Syntax Error in Command") as the reply to the last action it can parse.

8.2.3 TransactionPending

The receiver invokes the TransactionPending. A TransactionPending indicates that the transaction is actively being processed, but has not been completed. It is used to prevent the sender from assuming the TransactionRequest was lost where the transaction will take some time to complete.

TransactionPending(TransactionID { })

The TransactionID parameter must be the same as that of the corresponding TransactionRequest. A property of the Root Termination (normalMGExecutionTime) is settable by the MGC to indicate the interval within which the MGC expects a response to any transaction from the MG (exclusive of network delay). Another property (normalMGCExecutionTime) is settable by the MGC to indicate the interval within which the MG should expect a response to any transaction from the MGC (exclusive of network delay). MGProvisionalResponseTimerValue indicates the time within which the MGC should expect a Pending Response from the MG if a transaction cannot be completed (initially set to normalMGExecutionTime plus network delay, but may be lowered). MGCProvisionalResponseTimerValue has the corresponding meaning to the MG. Senders may receive more than one TransactionPending for a command. If a duplicate request is received when pending, the responder may send a duplicate pending immediately, or continue waiting for its timer to trigger another TransactionPending.

A property of the Root Termination (MGOriginatedPendingLimit) is settable by the MGC to indicate the number of TransactionPendings that can be received from the MG. When the value expressed by this property is exceeded, the MG shall stop the transaction processing and send back a TransactionReply, otherwise the MGC can assume the transaction to be in error.

Another property of the Root Termination (MGCOriginatedPendingLimit) is settable by the MGC to indicate the number of TransactionPendings that can be received from the MGC. When the value expressed by this property is exceeded, the MGC shall stop the transaction processing and send back a TransactionReply otherwise the MG can assume the transaction to be in error.

The xxxOriginatedPendingLimit (MGOriginatedPendingLimit or MGCOriginatedPendingLimit) may be exceeded either because of long command processing or due to an error (e.g., a command caused a loop). In both cases, the receiver of the original TransactionRequest will issue a TransactionReply with an Error Descriptor as a response parameter in correspondence with either the offending long command or the command that caused the error. Further commands in the transaction shall not be processed. Error 506 (Number of TransactionPendings Exceeded) shall be used.

NOTE – To prevent a situation where the xxxOriginatedPendingLimit (MGOriginatedPendingLimit or MGCOriginatedPendingLimit) is exceeded due to an error and the receiver of the original TransactionRequest keeps sending TransactionPending, the receiver of the original TransactionRequest should implement a management protection mechanism in order to trigger the appropriate recovery actions. The sender of the original TransactionRequest may keep track of the number of received Pendings and initiate corrective actions.

8.3 Messages

Multiple transactions can be concatenated into a Message. Messages have a header, which includes the identity of the sender. The Message Identifier (MID) of a message is set to a provisioned name (e.g., domain address/domain name/device name) of the entity transmitting the message. Domain name is a suggested default. An H.248.1 entity (MG/MGC) must consistently use the same MID in all messages it originates for the duration of control association with the peer (MGC/MG).

Every message contains a version number identifying the version of the protocol the message conforms to. Versions consist of one or two digits, beginning with version 1. The current version of the protocol is version 3.

The transactions in a message are treated independently. There is no order implied; there is no application or protocol acknowledgement of a message. A message is essentially a transport mechanism. For example, message X containing transaction requests A, B, and C may be responded to with message Y containing replies to A and C and message Z containing the reply to B. Likewise, message L containing request D and message M containing request E may be responded to with message N containing replies to both D and E.

9 Transport

The transport mechanism for the protocol should allow the reliable transport of transactions between a MGC and MG. The transport shall remain independent of what particular commands are being sent, and shall be applicable to all application states. There are several transports defined for the protocol, which are defined in annexes to this Recommendation and other H.248 Subseries Recommendations (e.g., H.248.4 and H.248.5). Additional transports may be defined as additional Recommendations in the H.248 subseries Recommendations. For transport of the protocol over IP, MGCs shall implement both TCP and UDP/ALF, a MG shall implement TCP or UDP/ALF or both.

The MG is provisioned with a name or address (such as a DNS name or IP address) of a primary and zero or more secondary MGCs (see 7.2.8) that is the address the MG uses to send messages to the MGC. If TCP or UDP is used as the protocol transport, and the port to which the initial ServiceChange request is to be sent is not otherwise known, that request should be sent to the default port number for the protocol. This port number is 2944 for text-encoded operation or 2945 for binary-encoded operation, for either UDP or TCP. The MGC receives the message containing the ServiceChange request from the MG and can determine the MG's address from it. As described in 7.2.8, either the MG or the MGC may supply an address in the ServiceChangeAddress parameter to which subsequent transaction requests must be addressed, but responses (including the response to the initial ServiceChange request) must always be sent back to the address which was the source of the corresponding request. For example, in IP networks, this is the source address in the IP header and the source port number in the TCP/UDP/SCTP header.

9.1 Ordering of commands

This Recommendation does not mandate that the underlying transport protocol guarantees the sequencing of transactions sent to an entity. This property tends to maximize the timeliness of actions, but it has a few drawbacks. For example:

- Notify commands may be delayed and arrive at the MGC after the transmission of a new command changing the Events Descriptor.
- If a new command is transmitted before a previous one is acknowledged, there is no guarantee that prior command will be executed before the new one.

Media Gateway Controllers that want to guarantee consistent operation of the Media Gateway may use the following rules. These rules are with respect to commands that are in different transactions. Commands that are in the same transaction are executed in order (see clause 8).

1) When a Media Gateway handles several terminations, commands pertaining to the different terminations may be sent in parallel, for example following a model where each termination (or group of terminations) is controlled by its own process or its own thread.

2) On a termination, there should normally be, at most, one outstanding command (Add or Modify or Move), unless the outstanding commands are in the same transaction. However, a Subtract command may be issued at any time. In consequence, a Media Gateway may

sometimes receive a Modify command that applies to a previously subtracted termination. Such commands should be ignored, and an error code should be returned.

3) For transports that do not guarantee in-sequence delivery of messages (i.e., UDP), there should normally be, on a given termination, at most one outstanding Notify command at any time.

4) In some cases, an implicitly, or explicitly, wildcarded Subtract Command that applies to a group of terminations may step in front of a pending Add Command. The Media Gateway Controller should individually delete all terminations for which an Add Command was pending at the time of the global Subtract Command. Also, new Add Commands for terminations named by the wildcarding (or implied in a Multiplex Descriptor) should not be sent until the wildcarded Subtract Command is acknowledged.

5) AuditValue and AuditCapability are not subject to any sequencing.

6) ServiceChange shall always be the first command sent by a MG as defined by the restart procedure. Any other command or response must be delivered after this ServiceChange Command.

These rules do not affect the command responder, which should always respond to commands.

9.2 Protection against restart avalanche

In the event that a large number of Media Gateways are powered on simultaneously, and they were to all initiate a ServiceChange transaction, the Media Gateway Controller would very likely be swamped, leading to message losses and network congestion during the critical period of service restoration. In order to prevent such avalanches, the following behaviour is suggested:

1) When a Media Gateway is powered on, it should initiate a restart timer to a random value, uniformly distributed between zero and a maximum waiting delay (MWD). Care should be taken to avoid synchronicity of the random number generation between multiple Media Gateways that would use the same algorithm.

2) The Media Gateway should then wait for either the end of this timer or the detection of a local user activity, such as for example an off-hook transition on a residential Media Gateway.

3) When the timer elapses, or when an activity is detected, the Media Gateway should initiate the restart procedure.

The restart procedure simply requires the MG to guarantee that the first message that the Media Gateway Controller sees from this MG is a ServiceChange message informing the Media Gateway Controller about the restart.

NOTE – The value of MWD is a configuration parameter that depends on the type of the Media Gateway. The following reasoning may be used to determine the value of this delay on residential gateways.

Media Gateway Controllers are typically dimensioned to handle the peak hour traffic load, during which, on average, 10% of the lines will be busy, placing calls whose average duration is typically three minutes. The processing of a call typically involves five to six Media Gateway Controller transactions between each Media Gateway and the Media Gateway Controller. This simple calculation shows that the Media Gateway Controller is expected to handle five to six transactions for each termination, every 30 minutes on average, or, to put it otherwise, about one transaction per termination every five to six minutes on average. This suggests that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, residential gateways should adopt a value of 600 seconds for MWD.

The same reasoning suggests that the value of MWD should be much shorter for trunking gateways or for business gateways, because they handle a large number of terminations, and also because the usage rate of these terminations is much higher than 10% during the peak busy hour, a typical value

being 60%. These terminations, during the peak hour, are thus expected to contribute about one transaction per minute to the Media Gateway Controller load. A reasonable algorithm is to make the value of MWD per "trunk" termination six times shorter than the MWD per residential gateway, and also inversely proportional to the number of terminations that are being restarted. For example, MWD should be set to 2.5 seconds for a gateway that handles a T1 line, or to 60 milliseconds for a gateway that handles a T3 line.

9.3 Protection against Notify avalanche

If a circumstance arises at an MG where a substantial number of notifications accumulate, either because of transmission difficulties or because the MG recognized a number of events in a short time period, the MG should send the notifications in a restricted manner until the backlog is cleared.

10 Security considerations

This clause covers security when using the protocol in an IP environment.

10.1 Protection of protocol connections

A security mechanism is clearly needed to prevent unauthorized entities from using the protocol defined in this Recommendation for setting up unauthorized calls or interfering with authorized calls. The security mechanism for the protocol when transported over IP networks is IPSec (RFC 2401 to RFC 2411).

The AH header (RFC 2402) affords data origin authentication, connectionless integrity and optional anti-replay protection of messages passed between the MG and the MGC. The ESP header (RFC 2406) provides confidentiality of messages, if desired. For instance, the ESP encryption service should be requested if the session descriptions are used to carry session keys, as defined in SDP.

Implementations of the protocol defined in this Recommendation employing the ESP header shall comply with section 5 of RFC 2406, which defines a minimum set of algorithms for integrity checking and encryption. Similarly, implementations employing the AH header shall comply with section 5 of RFC 2402, which defines a minimum set of algorithms for integrity checking using manual keys.

Implementations should use IKE (RFC 2409) to permit more robust keying options. Implementations employing IKE should support authentication with RSA signatures and RSA public key encryption.

10.2 Interim AH scheme

Implementation of IPSec requires that the AH or ESP header be inserted immediately after the IP header. This cannot be easily done at the application level. Therefore, this presents a deployment problem for the protocol defined in this Recommendation where the underlying network implementation does not support IPSec.

As an interim solution, an optional AH header is defined within the H.248.1 protocol header. The header fields are exactly those of the SPI, SEQUENCE NUMBER and DATA fields as defined in RFC 2402. The semantics of the header fields are the same as the "transport mode" of RFC 2402, except for the calculation of the Integrity Check Value (ICV). In IPSec, the ICV is calculated over the entire IP packet including the IP header. This prevents spoofing of the IP addresses. To retain the same functionality, the ICV calculation should be performed across all the transactions (concatenated) in the message prepended by a synthesized IP header consisting of a 32-bit source IP address, a 32-bit destination address and a 16-bit UDP destination port encoded as 20 hex digits. When the interim AH mechanism is employed when TCP is the transport Layer, the UDP Port above becomes the TCP port, and all other operations are the same.

Implementations of the H.248.1 protocol shall implement IPSec where the underlying operating system and the transport network supports IPSec. Implementations of the protocol using IPv4 shall implement the interim AH scheme. However, this interim scheme shall not be used when the underlying network layer supports IPSec. IPv6 implementations are assumed to support IPSec and shall not use the interim AH scheme.

All implementations of the interim AH mechanism shall comply with section 5 of RFC 2402 which defines a minimum set of algorithms for integrity checking using manual keys.

The interim AH interim scheme does not provide protection against eavesdropping, thus forbidding third parties from monitoring the connections set up by a given termination. Also, it does not provide protection against replay attacks. These procedures do not necessarily protect against denial of service attacks by misbehaving MGs or misbehaving MGCs. However, they will provide an identification of these misbehaving entities, which should then be deprived of their authorization through maintenance procedures.

10.3 Protection of media connections

The protocol allows the MGC to provide MGs with "session keys" that can be used to encrypt the audio messages, protecting against eavesdropping.

A specific problem of packet networks is "uncontrolled barge-in". This attack can be performed by directing media packets to the IP address and UDP port used by a connection. If no protection is implemented, the packets must be decompressed and the signals must be played on the "line side".

A basic protection against this attack is to only accept packets from known sources, checking for example that the IP source address and UDP source port match the values announced in the Remote Descriptor. This has two inconveniences: it slows down connection establishment and it can be fooled by source spoofing:

- To enable the address-based protection, the MGC must obtain the remote session description of the egress MG and pass it to the ingress MG. This requires at least one network round trip, and leaves us with a dilemma: allow the call to proceed without waiting for the round trip to complete and risk, for example, "clipping" a remote announcement, or wait for the full round trip and settle for slower call-set up procedures.
- Source spoofing is only effective if the attacker can obtain valid pairs of source destination addresses and ports, for example, by listening to a fraction of the traffic. To fight source spoofing, one could try to control all access points to the network. But this is in practice very hard to achieve.

An alternative to checking the source address is to encrypt and authenticate the packets, using a secret key that is conveyed during the call set-up procedure. This will not slow down the call set-up, and provides strong protection against address spoofing.

11 MG-MGC control interface

The control association between MG and MGC is initiated at MG cold start, and announced by a ServiceChange message, but can be changed by subsequent events, such as failures or manual service events.

NOTE – While the protocol does not have an explicit mechanism to support multiple MGCs controlling a physical MG, it has been designed to support the multiple logical MGs (within a single physical MG) that can be associated with different MGCs.

11.1 Multiple Virtual MGs

A physical Media Gateway may be partitioned into one or more Virtual MGs (VMGs). A Virtual MG consists of a set of statically partitioned physical terminations and/or sets of ephemeral

terminations. A physical termination is controlled by one MGC. The model does not require that other resources be statically allocated, just terminations. The mechanism for allocating terminations to Virtual MGs is a management method outside the scope of this Recommendation. Each of the Virtual MGs appears to the MGC as a complete MG client.

A physical MG may have only one network interface, which must be shared across Virtual MGs. In such a case, the packet/cell side termination is shared. It should be noted however, that in use, such interfaces require an ephemeral instance of the termination to be created per flow, and thus sharing the termination is straightforward. This mechanism does lead to a complication, namely that the MG must always know which of its controlling MGCs should be notified if an event occurs on the interface.

In normal operation, the Virtual MG will be instructed by the MGC to create network flows (if it is the originating side), or to expect flow requests (if it is the terminating side), and no confusion will arise. However, if an unexpected event occurs, the Virtual MG must know what to do with respect to the physical resources it is controlling.

If recovering from the event requires manipulation of a physical interface's state, only one MGC should do so. These issues are resolved by allowing any of the MGCs to create Events Descriptors to be notified of such events, but only one MGC can have read/write access to the physical interface properties; all other MGCs have read-only access. The management mechanism is used to designate which MGC has read/write capability, and is designated the Master MGC.

Each Virtual MG has its own Root Termination. In most cases, the values for the properties of the Root Termination are independently settable by each MGC. Where there can only be one value, the parameter is read-only to all but the Master MGC.

ServiceChange may only be applied to a termination or set of terminations partitioned to the Virtual MG or created (in the case of ephemeral terminations) by that Virtual MG.

11.2 Cold start

A MG is pre-provisioned by a management mechanism outside the scope of this Recommendation with a primary and (optionally) an ordered list of secondary MGCs. Upon a cold start of the MG, it will issue a ServiceChange Command with a "Restart" method, on the Root Termination to its primary MGC. If the MGC accepts the MG, it sends a TransactionReply not including a ServiceChangeMgcID parameter. If the MGC does not accept the MG's registration, it sends a TransactionReply, providing the address of an alternate MGC to be contacted by including a ServiceChangeMgcID parameter.

If the MG receives a TransactionReply that includes a ServiceChangeMgcID parameter, it sends a ServiceChange to the MGC specified in the ServiceChangeMgcID. It continues this process until it gets a controlling MGC to accept its registration, or it fails to get a reply. Upon failure to obtain a reply, either from the primary MGC, or a designated successor, the MG tries its pre-provisioned secondary MGCs, in order. If the MG is unable to establish a control relationship with any MGC, it shall wait a random amount of time as described in 9.2 and then start contacting its primary, and if necessary, its secondary MGCs again.

It is possible that the reply to a ServiceChange with Restart will be lost, and a command will be received by the MG prior to the receipt of the ServiceChange response. The MG shall issue Error Code 505 ("TransactionRequest Received before a ServiceChange Reply has been received").

11.3 Negotiation of protocol version

A ServiceChange Command from a MG that registers with an MGC shall contain the version number of the protocol supported by the MG in the ServiceChangeVersion parameter. Regardless of the version placed in the ServiceChangeVersion parameter, the message containing the command shall be encoded as a version 1 message. Upon receiving such a message, if the MGC supports only

a lower version, then the MGC shall send a ServiceChange Reply with the lower version and, thereafter, all the messages between MG and MGC shall conform to the lower version of the protocol. If the MG is unable to comply, and it has established a transport connection to the MGC, it should close that connection. In any event, it should reject all subsequent requests from the MGC with Error Code 406 ("Version Not Supported").

If the MGC only supports higher version(s) than the MG, it shall reject the association with Error Code 406 ("Version Not Supported").

If the MGC supports the version indicated by the MG, it shall conform to that version in all subsequent messages. In this case it is optional for the MGC to return a version in the ServiceChange Reply.

Protocol version negotiation may also occur at "Handoff" and "Failover" ServiceChanges.

When extending the protocol with new versions, the following rules should be followed:

1) Existing protocol elements, i.e., procedures, parameters, descriptor, property, values, should not be changed unless a protocol error needs to be corrected or it becomes necessary to change the operation of the service that is being supported by the protocol.
2) The semantics of a command, a parameter, a descriptor, a property, or a value should not be changed.
3) Established rules for formatting and encoding messages and parameters should not be modified.
4) When information elements are found to be obsolete they can be marked as not used. However, the identifier for that information element will be marked as reserved. In that way it cannot be used in future versions.

11.4 Failure of a MG

If a MG fails, but is capable of sending a message to the MGC, it sends a ServiceChange with an appropriate method (Graceful or Forced) and specifies the Root TerminationID. When it returns to service, it sends a ServiceChange with a "Restart" ServiceChangeMethod.

Allowing the MGC to send duplicate messages to both MGs accommodates pairs of MGs that are capable of redundant failover of one of the MGs. Only the Working MG shall accept or reject transactions. Upon failover, the primary MG sends a ServiceChange Command with a "Failover" method and a "MG Impending Failure" reason. The MGC then uses the secondary MG as the active MG. When the error condition is repaired, the working MG can send a "ServiceChange" with a "Restart" ServiceChangeMethod.

NOTE – Redundant failover MGs require a reliable transport, because the protocol provides no means for a secondary MG running ALF to acknowledge messages sent from the MGC.

11.5 Failure of an MGC

If the MG detects a failure of its controlling MGC, it attempts to contact the next MGC on its pre-provisioned list. It starts its attempts at the beginning (primary MGC), unless that was the MGC that failed, in which case it starts at its first secondary MGC. It sends a ServiceChange message with a "Failover" method and a "MGC Impending Failure" reason. If the MG is unable to establish a control relationship with any MGC, it shall wait a random amount of time as described in 9.2 and then start again contacting its primary, and (if necessary) its secondary MGCs. When contacting its previously controlling MGC, the MG sends the ServiceChange message with "Disconnected" method.

In partial failure, or for manual maintenance reasons, an MGC may wish to direct its controlled MGs to use a different MGC. To do so, it sends a ServiceChange method to the MG with a "Handoff" method, and its designated replacement in ServiceChangeMgcID. If "Handoff" is

supported, the MG shall send a ServiceChange message with a "Handoff" method and a "MGC directed change" reason to the designated MGC. If it fails to get a reply from the designated MGC, the MG shall behave as if its MGC failed, and start contacting secondary MGCs as specified in the previous paragraph. If the MG is unable to establish a control relationship with any MGC, it shall wait a random amount of time as described in 9.2 and then start contacting its primary, and if necessary, its secondary MGCs again.

No recommendation is made on how the MGCs involved in the handoff maintain state information; this is considered to be out of scope of this Recommendation. The MGC and MG may take the following steps when handoff occurs. When the MGC initiates a handoff, the handover should be transparent to operations on the Media Gateway. Transactions can be executed in any order, and could be in progress when the ServiceChange is executed. Accordingly, commands in progress continue and replies to all commands from the original MGC must be sent to the transport address from which they were sent. If the service relationship with the sending MGC has ended, the replies should be discarded. The MG may receive outstanding transaction replies from the new MGC. No new messages shall be sent to the new MGC until the control association is established. Repeated transaction requests shall be directed to the new MGC. The MG shall maintain the state of all terminations and contexts.

It is possible that the MGC could be implemented in such a way that a failed MGC is replaced by a working MGC where the identity of the new MGC is the same as the failed one. In such a case, the ServiceChangeMgcID would be specified with the previous value and the MG shall behave as if the value was changed, and send a ServiceChange message, as above.

Pairs of MGCs that are capable of redundant failover can notify the controlled MGs of the failover by the above mechanism.

11.6 MGC-MG control association monitoring

Monitoring the MGC-MG control association is essential for high availability networks. This may be realized by continuously checking the MGC-MG interconnection (link state). Many transports provide this functionality, so a mandatory protocol mechanism is unnecessary.

For those transports that do not provide link state monitoring, this functionality can be achieved in the H.248 protocol by utilizing the existing messaging. In the absence of any H.248 messaging, the MGC may use an AuditValue Command on Root with an empty Audit Descriptor to detect the loss of communications with the MG. The MG can detect the communications loss itself through the use of the Inactivity Timer Package (ITU-T Rec. H.248.14).

12 Package definition

The primary mechanism for extension is by means of Packages. Packages define additional properties that may occur on terminations and contexts and events, signals and statistics that may occur on terminations.

Packages defined by IETF will appear in separate RFCs.

Packages defined by ITU-T may appear in the relevant Recommendations (e.g., as in the H.248 subseries of Recommendations).

1) A public document or a standard forum document, which can be referenced as the document that describes the package following the guideline above, should be specified.
2) The document shall specify the version of the Package that it describes.
3) The document should be available on a public web server and should have a stable URL. The site should provide a mechanism to provide comments and appropriate responses should be returned.

12.1 Guidelines for defining packages

Packages define properties, events, signals, and statistics.

Packages may also define new error codes according to the guidelines given in 14.2. This is a matter of documentary convenience: the package documentation is submitted to IANA in support of the error code registration. If a package is modified, it is unnecessary to provide IANA with a new document reference in support of the error code unless the description of the error code itself is modified.

Names of all such defined constructs shall consist of the PackageID (which uniquely identifies the package) and the ID of the item (which uniquely identifies the item in that package). In the text encoding the two shall be separated by a forward slash ("/") character. Example: tonegen/playtone is the text encoding to refer to the play tone signal in the tone generation package.

A Package shall contain the following sections and keywords (indicated in bold). A template for defining packages is contained in Appendix II.

12.1.1 Package

Overall description of the package, specifying:

Package Name: only descriptive

PackageID: is an identifier

Description: is a description of the package

Version:

A new version of a package can only add additional properties, events, signals, statistics and new possible values for an existing parameter described in the original package. No deletions or modifications shall be allowed. A version is an integer in the range from 1 to 99.

Designed to be extended only (Optional): Yes

This indicates that the package has been expressly designed to be extended by others, not to be directly referenced. For example, the package may not have any function on its own or be nonsensical on its own. The MG shall not publish this PackageID when reporting packages.

Extends: existing package Identifier and version

A package may extend an existing package. The version of the original package must be specified. When a package extends another package it shall only add additional properties, events, signals, statistics and new possible values for an existing parameter described in the original package. An extended package shall not redefine or overload an identifier defined in the original package and packages it may have extended (multiple levels of extension). Hence, if package B version 1 extends package A version 1, version 2 of B will not be able to extend the A version 2 if A version 2 defines a name already in B version 1. If the package does not extend another package, it shall specify "none".

12.1.2 Properties

Properties defined by the package, specifying:

Property Name: only descriptive

PropertyID: is an identifier.

Description: is a description of the function of the property

Type: One of:

Boolean

String: UTF-8 string

Octet String: A number of octets. See Annex A and B.3 for encoding

Integer: 4-byte signed integer

Double: 8-byte signed integer

Character: Unicode UTF-8 encoding of a single letter. Could be more than one octet.

Enumeration: one of a list of possible unique values.

Sub-list: a list of several values from a list. The type of sub-list shall also be specified. The type shall be chosen from the types specified in this section (with the exception of sub-list). For example, Type: sub-list of enumeration. The encoding of sub-lists is specified in Annexes A and in B.2.

Possible values:

A package must specify either a specific set of values or a description of how values are determined. A package must also specify a default value or the default behaviour when the value is omitted from its descriptor. For example, a package may specify that procedures related to the property are suspended when its value is omitted.

Default:

A default value (but not procedures) may be specified as provisionable.

Defined in:

The H.248.1 descriptor in which the property is defined. LocalControl is for stream-dependent properties. TerminationState is for stream-independent properties. ContextAttribute is for properties that affect the context as a whole, i.e., mixing properties. These are expected to be the most common cases, but it is possible for properties to be defined in other descriptors. Context properties must be defined in the ContextAttribute Descriptor.

Characteristics: Read-Only or Read/Write, and (optionally), global:

Indicates whether a property is read-only, or read/write, and if it is global. If global is omitted, the property is not global. If a property is declared as global, the value of the property is shared by all terminations realizing the package. If a context property is declared as global, the property is shared by all contexts realizing the package.

12.1.3 Events

Events defined by the package, specifying:

Event name: only descriptive

EventID: is an identifier

Description: a description of the function of the event

EventsDescriptor Parameters:

Parameters used by the MGC to configure the event, and found in the Events Descriptor. See 12.2. If there are no parameters for the Events Descriptor, then "none" shall be specified.

ObservedEventsDescriptor Parameters:

Parameters returned to the MGC in Notify requests and in replies to command requests from the MGC that audit ObservedEvents Descriptor, and found in the ObservedEvents

Descriptor. See 12.2. If there are no parameters for the ObservedEvents Descriptor, then "none" shall be specified.

12.1.4 Signals

Signals defined by the package, specifying:

Signal Name: only descriptive

SignalID: is an identifier. SignalID is used in a Signals Descriptor

Description: a description of the function of the signal

SignalType: one of:

OO (OnOff)

TO (TimeOut)

BR (Brief)

NOTE – SignalType may be defined such that it is dependent on the value of one or more parameters. The package must specify a default signal type. If the default type is TO, the package must specify a default duration which may be provisioned. A default duration is meaningless for BR.

Duration: in hundredths of seconds

Additional Parameters: see 12.2

12.1.5 Statistics

Statistics defined by the package, specifying:

Statistic name: only descriptive

StatisticID: is an identifier

StatisticID is used in a Statistics Descriptor

Description: a description of the statistic

Type: One of:

Boolean

String: UTF-8 string

Octet String: A number of octets. See Annex A and B.3 for encoding

Integer: 4-byte signed integer

Double: 8-byte signed integer

Character: Unicode UTF-8 encoding of a single letter. Could be more than one octet.

Enumeration: One of a list of possible unique values.

Sub-list: A list of several values from a list. The type of sub-list shall also be specified. The type shall be chosen from the types specified in this section (with the exception of sub-list). For example, Type: sub-list of enumeration. The encoding of sub-lists is specified in Annexes A and in B.2.

Possible values:

A package must indicate the unit of measure, e.g., milliseconds, packets, either here or along with the type above, as well as indicating any restriction on the range.

Level: Specify if the statistic can be kept at the termination level, Stream level or either.

12.1.6 Error Codes

If the package does not define any error codes, this section may be omitted. Otherwise, it describes error codes defined by the package, specifying:

Error Code #: The error code number.

Name: Name of the error

Definition: A description of the error code.

Error Text in the Error Descriptor:

A description of what text to return in the Error Descriptor.

Comment: Any further comments on the use of the error code.

12.1.7 Procedures

Additional guidance on the use of the package.

12.2 Guidelines to defining parameters to events and signals

Parameter Name: only descriptive

ParameterID: is an identifier. The textual ParameterID of parameters to events and signals shall not start with "EPA" and "SPA", respectively. The textual ParameterID shall also not be "ST", "Stream", "SY", "SignalType", "DR", "Duration", "NC", "NotifyCompletion", "KA", "KeepActive", "EB", "Embed", "DM", "DigitMap", "SPADI", "SPADirection", "SPARQ" or "SPARequestID".

Description: a description of the function of the parameter.

Type: One of:

Boolean

String: UTF-8 octet string

Octet String: A number of octets. See Annex A and B.3 for encoding

Integer: 4-octet signed integer

Double: 8-octet signed integer

Character: Unicode UTF-8 encoding of a single letter. Could be more than one octet.

Enumeration: one of a list of possible unique values.

Sub-list: a list of several values from a list (not supported for statistics). The type of sub-list shall also be specified. The type shall be chosen from the types specified in this section (with the exception of sub-list). For example, Type: sub-list of enumeration. The encoding of sub-lists is specified in Annex A and B.2.

Optional: Yes/No

Describes if the parameter may be omitted from the signal or event.

Possible values:

A package must specify either a specific set of values or a description of how values are determined. A package must also specify a default value or the default behaviour when the value is omitted from its descriptor. For example, a package may specify that procedures related to the parameter are suspended when its value is omitted.

Default:

A default value (but not procedures) may be specified as provisionable.

12.3 Identifiers

Identifiers in text encoding shall be strings of up to 64 characters, containing no spaces, starting with an alphabetic character and consisting of alphanumeric characters and/or digits, and possibly including the special character underscore ("_").

Identifiers in binary encoding are 2 octets long.

Both text and binary values shall be specified for each identifier, including identifiers used as values in enumerated types.

12.4 Package registration

A package can be registered with IANA for interoperability reasons. See clause 14 for IANA considerations.

13 Profile definition

Profiles may be specified to further define how the H.248.1 protocol is used and what functionality is supported by a MG. It only describes the capabilities of the MGC/MG H.248 interface. The profile itself specifies what options associated with H.248.1 have been used. For example: transport and packages used for an application.

A profile is identified by a name (IANA registered) and a version. A name shall be a case-insensitive string up to 64 characters long. Version shall be 1 to 99.

The profile itself is a document that indicates the options for a particular application. There is no set format for this document. The only mandatory element is that there should be a section indicating the Profile Name and Version and a summary of the profile.

The two points below are the only mandatory sections of a profile:

- Profile Identification: The name and version of the profile that is sent in the ServiceChange Command.
- Summary: A description of what the profile is.

Appendix III contains a template for the definition of profiles. It should be used as the basis of a profile definition.

14 IANA considerations

14.1 Packages

The following considerations shall be met to register a package with IANA:

1) A unique string name, unique serial number and version number is registered for each package. The string name is used with text encoding. The serial number shall be used with binary encoding. Serial Numbers 0x8000 to 0xFFFF are reserved for private use. Serial number 0 is reserved.
2) A contact name, email and postal addresses for that contact shall be specified. The contact information shall be updated by the defining organization as necessary.
3) A reference to a document that describes the package, which should be public:

   The document shall specify the version of the package that it describes.

   If the document is public, it should be located on a public web server and should have a stable URL. The site should provide a mechanism to provide comments and appropriate responses should be returned.

4) Packages registered by other than recognized standards bodies shall have a minimum package name length of 8 characters.

5) All other package names are first come-first served if all other conditions are met.

14.2 Error codes

The following considerations shall be met to register an error code with IANA:

1) An error number and a one-line (80-character maximum) string are registered for each error.

2) A complete description of the conditions under which the error is detected shall be included in a publicly available document. The description shall be sufficiently clear to differentiate the error from all other existing error codes.

3) The document should be available on a public web server and should have a stable URL.

4) Error numbers registered by recognized standards bodies shall have 3- or 4-character error numbers.

5) Error numbers registered by all other organizations or individuals shall have 4-character error numbers.

6) An error number shall not be redefined nor modified except by the organization or individual that originally defined it, or their successors or assigns.

14.3 ServiceChange reasons

The following considerations shall be met to register ServiceChange reason with IANA:

1) A one-phrase, 80-character maximum, unique reason code is registered for each reason.

2) A complete description of the conditions under which the reason is used shall be included in a publicly available document. The description shall be sufficiently clear to differentiate the reason from all other existing reasons.

3) The document should be available on a public web server and should have a stable URL.

14.4 Profiles

The following considerations shall be met to register a profile with IANA:

1) A unique string name and version number (version may be omitted when the profile name contains a wildcard) is registered for each profile.

2) A contact name, email and postal addresses for that contact shall be specified. The contact information shall be updated by the defining organization as necessary.

3) Profiles registered by other than recognized standards bodies shall have a minimum profile name length of 6 characters.

4) Profile names containing a wildcard "*" on the end of their names shall be accepted if the first 6 characters are fully specified. It is assumed that the organization that was issued with the profile name will manage the namespace associated with the wildcard. IANA shall not issue other profiles names within "name*" range.

All other profile names are first come-first served if all other conditions are met.

Annex A

Binary encoding of the protocol

This annex specifies the syntax of messages using the notation defined in ITU-T Rec. X.680, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.* Messages shall be encoded for transmission by applying the basic encoding rules specified in ITU-T Rec. X.690, *Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.

A.1 Coding of wildcards

The use of wildcards ALL and CHOOSE is allowed in the protocol. This allows a MGC to partially specify TerminationIDs and to let the MG choose from the values that conform to the partial specification. TerminationIDs may encode a hierarchy of names. This hierarchy is provisioned. For instance, a TerminationID may consist of a trunk group, a trunk within the group and a circuit. Wildcarding must be possible at all levels. The following paragraphs explain how this is achieved.

The ASN.1 description uses octet strings of up to 8 octets in length for TerminationIDs. This means that TerminationIDs consist of at most 64 bits. A fully specified TerminationID may be preceded by a sequence of wildcarding fields. A wildcarding field is one octet in length. Bit 7 (the most significant bit) of this octet specifies what type of wildcarding is invoked: if the bit value equals 1, then the ALL wildcard is used; if the bit value if 0, then the CHOOSE wildcard is used. Bit 6 of the wildcarding field specifies whether the wildcarding pertains to one level in the hierarchical naming scheme (bit value 0) or to the level of the hierarchy specified in the wildcarding field plus all lower levels (bit value 1). Bits 0 through 5 of the wildcarding field specify the bit position in the TerminationID at which the wildcarding starts.

We illustrate this scheme with some examples. In these examples, the most significant bit in a string of bits appears on the left-hand side.

Assume that TerminationIDs are three octets long and that each octet represents a level in a hierarchical naming scheme. A valid TerminationID is:

00000001 00011110 01010101.

Addressing ALL names with prefix 00000001 00011110 is done as follows:

wildcarding field: 10000111

TerminationID: 00000001 00011110 xxxxxxxx.

The values of the bits labelled "x" is irrelevant and shall be ignored by the receiver.

Indicating to the receiver that it must choose a name with 00011110 as the second octet is done as follows:

wildcarding fields: 00010111 followed by 00000111

TerminationID: xxxxxxxx 00011110 xxxxxxxx.

The first wildcard field indicates a CHOOSE wildcard for the level in the naming hierarchy starting at bit 23, the highest level in our assumed naming scheme. The second wildcard field indicates a CHOOSE wildcard for the level in the naming hierarchy starting at bit 7, the lowest level in our assumed naming scheme.

Finally, a CHOOSE-wildcarded name with the highest level of the name equal to 00000001 is specified as follows:

wildcard field: 01001111

TerminationID: 0000001 xxxxxxxx xxxxxxxx.

Bit value 1 at bit position 6 of the first octet of the wildcard field indicates that the wildcarding pertains to the specified level in the naming hierarchy and all lower levels.

ContextIDs may also be wildcarded. In the case of ContextIDs, however, specifying partial names is not allowed. ContextID 0x0 shall be used to indicate the NULL Context, ContextID 0xFFFFFFFE shall be used to indicate a CHOOSE wildcard, and ContextID 0xFFFFFFFF shall be used to indicate an ALL wildcard.

TerminationID 0xFFFFFFFFFFFFFFFF shall be used to indicate the Root Termination.

A.2 ASN.1 syntax specification

This clause contains the ASN.1 specification of the H.248.1 protocol syntax.

NOTE 1 – In case a transport mechanism is used that employs application level framing, the definition of **Transaction** below changes. Refer to the annex or to the Recommendation of the H.248.x subseries defining the transport mechanism for the definition that applies in that case.

NOTE 2 – This syntax specification does not enforce all restrictions on element inclusions and values. Some additional restrictions are stated in comments and other restrictions appear in the text of this Recommendation. These additional restrictions are part of the protocol even though not enforced by this Recommendation.

NOTE 3 – The ASN.1 module in this annex uses octet string types to encode values for property parameter, signal parameter and event parameter values and statistics. The actual types of these values vary and are specified in Annex C or the relevant package definition.

A value is first BER-encoded based on its type using the table below. The result of this BER-encoding is then encoded as an ASN.1 octet string, "double wrapping" the value. The format specified in Annex C or the package relates to BER encoding according to the following table:

Type Specified in Package	ASN.1 BER Type
String	IA5String or UTF8String (Note 4)
Integer (4-octet)	INTEGER
Double (8-octet signed int)	INTEGER (Note 3)
Character (UTF-8) (Note 1)	IA5String
Enumeration	ENUMERATED
Boolean	BOOLEAN
Unsigned Integer (Note 2)	INTEGER (Note 3)
Octet (String)	OCTET STRING
NOTE 1 – Can be more than one byte. NOTE 2 – Unsigned integer is referenced in Annex C. NOTE 3 – The BER encoding of INTEGER does not imply the use of 4 bytes. NOTE 4 – String should be encoded as IA5String when the contents are all ASCII characters, but as UTF8String if it contains any non-ASCII characters.	

See 8.7/X.690, for the definition of the encoding of an octet string value.

```
MEDIA-GATEWAY-CONTROL {itu-t(0) recommendation(0) h(8) h248(248) modules(0)
media-gateway-control(0) version3(3)}
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
MegacoMessage      ::= SEQUENCE
{
     authHeader              AuthenticationHeader OPTIONAL,
     mess                    Message
}
```

```
AuthenticationHeader    ::= SEQUENCE
{
     secParmIndex            SecurityParmIndex,
     seqNum                  SequenceNum,
     ad                      AuthData
}

SecurityParmIndex  ::= OCTET STRING(SIZE(4))
SequenceNum        ::= OCTET STRING(SIZE(4))
AuthData           ::= OCTET STRING (SIZE (12..32))
Message            ::= SEQUENCE
{
     version                 INTEGER(0..99),
     -- The version of the protocol defined here is equal to 3.
     mId                     Mid, -- Name/address of message originator
     messageBody             CHOICE
     {
          messageError            ErrorDescriptor,
          transactions            SEQUENCE OF Transaction
     },
     ...
}

Mid                ::= CHOICE
{
     ip4Address              IP4Address,
     ip6Address              IP6Address,
     domainName              DomainName,
     deviceName              PathName,
     mtpAddress              OCTET STRING(SIZE(2..4)),
     -- Addressing structure of mtpAddress:
     --    25 - 15           0
     --       |  PC        | NI |
     --    24 - 14 bits    2 bits
     -- NOTE – 14 bits are defined for international use.
     -- Two national options exist where the point code is 16 or 24 bits.
     -- To octet align the mtpAddress, the MSBs shall be encoded as 0s.
     ...
}

DomainName         ::= SEQUENCE
{
     name                    IA5String,
     -- The name starts with an alphanumeric digit followed by a sequence
     -- of alphanumeric digits, hyphens and dots. No two dots shall occur
     -- consecutively.
     portNumber              INTEGER(0..65535) OPTIONAL
}

IP4Address              ::= SEQUENCE
{
     address                 OCTET STRING (SIZE(4)),
     portNumber              INTEGER(0..65535) OPTIONAL
}

IP6Address              ::= SEQUENCE
{
     address                 OCTET STRING (SIZE(16)),
     portNumber              INTEGER(0..65535) OPTIONAL
}

PathName           ::= IA5String(SIZE (1..64))
-- See A.3
```

```
Transaction ::= CHOICE
{
     transactionRequest          TransactionRequest,
     transactionPending          TransactionPending,
     transactionReply            TransactionReply,
     transactionResponseAck      TransactionResponseAck,
     -- use of response acks is dependent on underlying transport
     ...,
     segmentReply                SegmentReply
}

TransactionId      ::= INTEGER(0..4294967295)   -- 32-bit unsigned integer

TransactionRequest ::= SEQUENCE
{
     transactionId          TransactionId,
     actions                SEQUENCE OF ActionRequest,
     ...
}

TransactionPending ::= SEQUENCE
{
     transactionId          TransactionId,
     ...
}

TransactionReply        ::= SEQUENCE
{
     transactionId          TransactionId,
     immAckRequired         NULL OPTIONAL,
     transactionResult      CHOICE
     {
          transactionError       ErrorDescriptor,
          actionReplies          SEQUENCE OF ActionReply
     },
     ...,
     segmentNumber          SegmentNumber OPTIONAL,
     segmentationComplete   NULL OPTIONAL
}

SegmentReply       ::= SEQUENCE
{
     transactionId          TransactionId,
     segmentNumber          SegmentNumber,
     segmentationComplete   NULL OPTIONAL,
     ...
}

SegmentNumber      ::= INTEGER(0..65535)

TransactionResponseAck  ::= SEQUENCE OF TransactionAck
TransactionAck          ::= SEQUENCE
{
     firstAck               TransactionId,
     lastAck                TransactionId OPTIONAL
}

ErrorDescriptor         ::= SEQUENCE
{
     errorCode              ErrorCode,
     errorText              ErrorText OPTIONAL
}
```

```
ErrorCode           ::= INTEGER(0..65535)
-- See clause 14 for IANA considerations with respect to error codes

ErrorText           ::= IA5String

ContextID           ::= INTEGER(0..4294967295)
-- Context NULL Value: 0
-- Context CHOOSE Value: 4294967294 (0xFFFFFFFE)
-- Context ALL Value: 4294967295 (0xFFFFFFFF)

ActionRequest       ::= SEQUENCE
{
     contextId               ContextID,
     contextRequest          ContextRequest OPTIONAL,
     contextAttrAuditReq     ContextAttrAuditRequest OPTIONAL,
     commandRequests         SEQUENCE OF CommandRequest
}

ActionReply             ::= SEQUENCE
{
     contextId               ContextID,
     errorDescriptor         ErrorDescriptor OPTIONAL,
     contextReply            ContextRequest OPTIONAL,
     commandReply            SEQUENCE OF CommandReply
}

ContextRequest      ::= SEQUENCE
{
     priority                INTEGER(0..15) OPTIONAL,
     emergency               BOOLEAN OPTIONAL,
     topologyReq             SEQUENCE OF TopologyRequest OPTIONAL,
     ...,
     iepscallind             BOOLEAN OPTIONAL,
     contextProp             SEQUENCE OF PropertyParm OPTIONAL,
     contextList             SEQUENCE OF ContextID OPTIONAL
}
-- When returning a contextList, the contextId in the ActionReply construct will
-- return the contextId from the associated ActionRequest.

ContextAttrAuditRequest ::= SEQUENCE
{
     topology                NULL OPTIONAL,
     emergency               NULL OPTIONAL,
     priority                NULL OPTIONAL,
     ...,
     iepscallind             NULL OPTIONAL,
     contextPropAud          SEQUENCE OF IndAudPropertyParm OPTIONAL,
     selectpriority          INTEGER(0..15) OPTIONAL,
           -- to select given priority
     selectemergency         BOOLEAN OPTIONAL,
           -- to select if emergency set/not set (T/F)
     selectiepscallind       BOOLEAN OPTIONAL,
           -- to select if IEPS set/not set (T/F)
     selectLogic             SelectLogic OPTIONAL    -- default is AND
}

SelectLogic             ::= CHOICE
{
     andAUDITSelect      NULL,      -- all filter conditions satisfied
     orAUDITSelect       NULL,      -- at least one filter condition satisfied
     ...
}

CommandRequest      ::= SEQUENCE
```

```
{
     command                 Command,
     optional                NULL OPTIONAL,
     wildcardReturn          NULL OPTIONAL,
     ...
}

Command            ::= CHOICE
{
     addReq                  AmmRequest,
     moveReq                 AmmRequest,
     modReq                  AmmRequest,
     -- Add, Move, Modify requests have the same parameters
     subtractReq             SubtractRequest,
     auditCapRequest         AuditRequest,
     auditValueRequest       AuditRequest,
     notifyReq               NotifyRequest,
     serviceChangeReq        ServiceChangeRequest,
     ...
}

CommandReply       ::= CHOICE
{
     addReply                AmmsReply,
     moveReply               AmmsReply,
     modReply                AmmsReply,
     subtractReply           AmmsReply,
     -- Add, Move, Modify, Subtract replies have the same parameters
     auditCapReply           AuditReply,
     auditValueReply         AuditReply,
     notifyReply             NotifyReply,
     serviceChangeReply      ServiceChangeReply,
     ...
}
TopologyRequest         ::= SEQUENCE
{
     terminationFrom         TerminationID,
     terminationTo           TerminationID,
     topologyDirection       ENUMERATED
     {
          bothway(0),
          isolate(1),
          oneway(2)
     },
     ...,
     streamID                StreamID OPTIONAL,
     topologyDirectionExtension  ENUMERATED
     {
          onewayexternal(0),
          onewayboth(1),
          ...
     }
}

AmmRequest         ::= SEQUENCE
{
     terminationID           TerminationIDList,
     descriptors             SEQUENCE OF AmmDescriptor,
     -- At most one descriptor of each type (see AmmDescriptor)
     -- allowed in the sequence.
     ...
}

AmmDescriptor      ::= CHOICE
```

```
{
     mediaDescriptor         MediaDescriptor,
     modemDescriptor         ModemDescriptor,
     muxDescriptor           MuxDescriptor,
     eventsDescriptor        EventsDescriptor,
     eventBufferDescriptor   EventBufferDescriptor,
     signalsDescriptor       SignalsDescriptor,
     digitMapDescriptor      DigitMapDescriptor,
     auditDescriptor         AuditDescriptor,
     ...,
     statisticsDescriptor    StatisticsDescriptor
}

AmmsReply          ::= SEQUENCE
{
     terminationID           TerminationIDList,
     terminationAudit        TerminationAudit OPTIONAL,
     ...
}

SubtractRequest         ::= SEQUENCE
{
     terminationID           TerminationIDList,
     auditDescriptor         AuditDescriptor OPTIONAL,
     ...
}

AuditRequest       ::= SEQUENCE
{
     terminationID           TerminationID,
     auditDescriptor         AuditDescriptor,
     ...,
     terminationIDList       TerminationIDList OPTIONAL
}
-- terminationID shall contain the first termination in the
-- list when using the terminationIDList construct in AuditRequest

AuditReply              ::= CHOICE
{
     contextAuditResult          TerminationIDList,
     error                       ErrorDescriptor,
     auditResult                 AuditResult,
     ...,
     auditResultTermList         TermListAuditResult
}
AuditResult             ::= SEQUENCE
{
     terminationID               TerminationID,
     terminationAuditResult      TerminationAudit
}

TermListAuditResult     ::= SEQUENCE
{
     terminationIDList           TerminationIDList,
     terminationAuditResult      TerminationAudit,
     ...
}

TerminationAudit        ::= SEQUENCE OF AuditReturnParameter

AuditReturnParameter    ::= CHOICE
{
     errorDescriptor             ErrorDescriptor,
     mediaDescriptor             MediaDescriptor,
```

```
     modemDescriptor              ModemDescriptor,
     muxDescriptor                MuxDescriptor,
     eventsDescriptor             EventsDescriptor,
     eventBufferDescriptor        EventBufferDescriptor,
     signalsDescriptor            SignalsDescriptor,
     digitMapDescriptor           DigitMapDescriptor,
     observedEventsDescriptor     ObservedEventsDescriptor,
     statisticsDescriptor         StatisticsDescriptor,
     packagesDescriptor           PackagesDescriptor,
     emptyDescriptors             AuditDescriptor,
     ...
}

AuditDescriptor         ::= SEQUENCE
{
     auditToken                   BIT STRING
     {
          muxToken(0),
          modemToken(1),
          mediaToken(2),
          eventsToken(3),
          signalsToken(4),
          digitMapToken(5),
          statsToken(6),
          observedEventsToken(7),
          packagesToken(8),
          eventBufferToken(9)
     } OPTIONAL,
     ...,
     auditPropertyToken      SEQUENCE OF IndAuditParameter OPTIONAL
}

IndAuditParameter  ::= CHOICE
{
     indaudmediaDescriptor        IndAudMediaDescriptor,
     indaudeventsDescriptor       IndAudEventsDescriptor,
     indaudeventBufferDescriptor  IndAudEventBufferDescriptor,
     indaudsignalsDescriptor      IndAudSignalsDescriptor,
     indauddigitMapDescriptor     IndAudDigitMapDescriptor,
     indaudstatisticsDescriptor   IndAudStatisticsDescriptor,
     indaudpackagesDescriptor     IndAudPackagesDescriptor,
     ...
}

IndAudMediaDescriptor   ::= SEQUENCE
{
     termStateDescr          IndAudTerminationStateDescriptor OPTIONAL,
     streams                 CHOICE
     {
          oneStream               IndAudStreamParms,
          multiStream             SEQUENCE OF IndAudStreamDescriptor
     } OPTIONAL,
     ...
}

IndAudStreamDescriptor  ::= SEQUENCE
{
     streamID                IndAudStreamID,
     streamParms             IndAudStreamParms
}

IndAudStreamParms  ::= SEQUENCE
{
     localControlDescriptor      IndAudLocalControlDescriptor OPTIONAL,
```

```
    localDescriptor             IndAudLocalRemoteDescriptor OPTIONAL,
    remoteDescriptor            IndAudLocalRemoteDescriptor OPTIONAL,
    ...,
    statisticsDescriptor        IndAudStatisticsDescriptor OPTIONAL
}

IndAudLocalControlDescriptor ::= SEQUENCE
{
    streamMode              NULL OPTIONAL,
    reserveValue            NULL OPTIONAL,
    reserveGroup            NULL OPTIONAL,
    propertyParms           SEQUENCE OF IndAudPropertyParm OPTIONAL,
    ...,
    streamModeSel           StreamMode OPTIONAL
}
-- must not have both streamMode and streamModeSel
-- if both are present only streamModeSel shall be honoured

IndAudPropertyParm ::= SEQUENCE
{
    name                    PkgdName,
    ...,
    propertyParms           PropertyParm OPTIONAL
}

--  to select based on property values
--  AND/OR selection logic is specified at context level

IndAudLocalRemoteDescriptor ::= SEQUENCE
{
    propGroupID             INTEGER(0..65535) OPTIONAL,
    propGrps                IndAudPropertyGroup,
    ...
}

IndAudPropertyGroup     ::= SEQUENCE OF IndAudPropertyParm

IndAudTerminationStateDescriptor ::= SEQUENCE
{
    propertyParms           SEQUENCE OF IndAudPropertyParm,
    eventBufferControl      NULL OPTIONAL,
    serviceState            NULL OPTIONAL,
    ...,
    serviceStateSel         ServiceState OPTIONAL
}
-- must not have both serviceState and serviceStateSel
-- if both are present only serviceStateSel shall be honoured

IndAudEventsDescriptor  ::= SEQUENCE
{
    requestID               RequestID OPTIONAL,
    pkgdName                PkgdName,
    streamID                StreamID OPTIONAL,
    ...
}

IndAudEventBufferDescriptor ::= SEQUENCE
{
    eventName               PkgdName,
    streamID                StreamID OPTIONAL,
    ...
}
```

```
IndAudSignalsDescriptor ::=CHOICE
{
     signal                  IndAudSignal,
     seqSigList              IndAudSeqSigList,
     ...
}

IndAudSeqSigList        ::= SEQUENCE
{
     id                      INTEGER(0..65535),
     signalList              IndAudSignal OPTIONAL
}

IndAudSignal       ::= SEQUENCE
{
     signalName              PkgdName,
     streamID                StreamID OPTIONAL,
     ...,
     signalRequestID         RequestID OPTIONAL
}

IndAudDigitMapDescriptor ::= SEQUENCE
{
     digitMapName            DigitMapName OPTIONAL
}

IndAudStatisticsDescriptor ::= SEQUENCE
{
     statName                PkgdName
}

IndAudPackagesDescriptor ::= SEQUENCE
{
     packageName             Name,
     packageVersion          INTEGER(0..99),
     ...
}
NotifyRequest      ::= SEQUENCE
{
     terminationID               TerminationIDList,
     observedEventsDescriptor    ObservedEventsDescriptor,
     errorDescriptor             ErrorDescriptor OPTIONAL,
     ...
}

NotifyReply             ::= SEQUENCE
{
     terminationID           TerminationIDList,
     errorDescriptor         ErrorDescriptor OPTIONAL,
     ...
}

ObservedEventsDescriptor ::= SEQUENCE
{
     requestId               RequestID,
     observedEventLst        SEQUENCE OF ObservedEvent
}
ObservedEvent      ::= SEQUENCE
```

```
{
     eventName               EventName,
     streamID                StreamID OPTIONAL,
     eventParList            SEQUENCE OF EventParameter,
     timeNotation            TimeNotation OPTIONAL,
     ...
}

EventName          ::= PkgdName

EventParameter     ::= SEQUENCE
{
     eventParameterName      Name,
     value                   Value,
     -- For use of extraInfos see the comment related to PropertyParm
     extraInfo               CHOICE
     {
          relation                Relation,
          range                   BOOLEAN,
          sublist                 BOOLEAN
     } OPTIONAL,
     ...

}

ServiceChangeRequest    ::= SEQUENCE
{
     terminationID           TerminationIDList,
     serviceChangeParms      ServiceChangeParm,
     ...
}

ServiceChangeReply ::= SEQUENCE
{
     terminationID           TerminationIDList,
     serviceChangeResult     ServiceChangeResult,
     ...
}

-- For ServiceChangeResult, no parameters are mandatory. Hence the
-- distinction between ServiceChangeParm and ServiceChangeResParm.
ServiceChangeResult ::= CHOICE
{
     errorDescriptor             ErrorDescriptor,
     serviceChangeResParms       ServiceChangeResParm
}

WildcardField      ::= OCTET STRING(SIZE(1))

TerminationID      ::= SEQUENCE
{
     wildcard                SEQUENCE OF WildcardField,
     id                      OCTET STRING(SIZE(1..8)),
     ...
}
-- See A.1 for explanation of wildcarding mechanism.
-- TerminationID 0xFFFFFFFFFFFFFFFF indicates the Root Termination.

TerminationIDList       ::= SEQUENCE OF TerminationID
```

```
MediaDescriptor         ::= SEQUENCE
{

     termStateDescr          TerminationStateDescriptor OPTIONAL,
     streams                 CHOICE
     {
          oneStream               StreamParms,
          multiStream             SEQUENCE OF StreamDescriptor
     } OPTIONAL,
     ...
}

StreamDescriptor        ::= SEQUENCE
{
     streamID                StreamID,
     streamParms             StreamParms
}

StreamParms        ::= SEQUENCE
{
     localControlDescriptor      LocalControlDescriptor OPTIONAL,
     localDescriptor             LocalRemoteDescriptor OPTIONAL,
     remoteDescriptor            LocalRemoteDescriptor OPTIONAL,
     ...,
     statisticsDescriptor        StatisticsDescriptor OPTIONAL
}

LocalControlDescriptor  ::= SEQUENCE
{
     streamMode              StreamMode OPTIONAL,
     reserveValue            BOOLEAN OPTIONAL,
     reserveGroup            BOOLEAN OPTIONAL,
     propertyParms           SEQUENCE OF PropertyParm,
     ...
}

StreamMode              ::= ENUMERATED
{
     sendOnly(0),
     recvOnly(1),
     sendRecv(2),
     inactive(3),
     loopBack(4),
     ...
}

-- In PropertyParm, value is a SEQUENCE OF octet string. When sent
-- by an MGC the interpretation is as follows:
-- empty sequence means CHOOSE
-- one element sequence specifies value
-- If the sublist field is not selected, a longer sequence means
-- "choose one of the values" (i.e., value1 OR value2 OR ...)
-- If the sublist field is selected,
-- a sequence with more than one element encodes the value of a
-- list-valued property (i.e., value1 AND value2 AND ...).
-- The relation field may only be selected if the value sequence
-- has length 1.  It indicates that the MG has to choose a value
-- for the property. E.g., x > 3 (using the greaterThan
-- value for relation) instructs the MG to choose any value larger
-- than 3 for property x.
-- The range field may only be selected if the value sequence
-- has length 2.  It indicates that the MG has to choose a value
-- in the range between the first octet in the value sequence and
-- the trailing octet in the value sequence, including the
```

```
-- boundary values.
-- When sent by the MG, only responses to an AuditCapability request
-- may contain multiple values, a range, or a relation field.

PropertyParm       ::= SEQUENCE
{
     name                    PkgdName,
     value                   SEQUENCE OF OCTET STRING,
     extraInfo               CHOICE
     {
          relation                Relation,
          range                   BOOLEAN,
          sublist                 BOOLEAN
     } OPTIONAL,
     ...
}

Name               ::= OCTET STRING(SIZE(2))

PkgdName           ::= OCTET STRING(SIZE(4))
-- represents Package Name (2 octets) plus property, event,
-- signal names or StatisticsID. (2 octets)
-- To wildcard a package use 0xFFFF for first two octets, CHOOSE
-- is not allowed. To reference native property tag specified in
-- Annex C, use 0x0000 as first two octets.
-- To wildcard a PropertyID, EventID, SignalID, or StatisticsID, use
-- 0xFFFF for last two octets, CHOOSE is not allowed.
-- Wildcarding of Package Name is permitted only if PropertyID,
-- EventID, SignalID, or StatisticsID are also wildcarded.

Relation           ::= ENUMERATED
{
     greaterThan(0),
     smallerThan(1),
     unequalTo(2),
     ...}

LocalRemoteDescriptor   ::= SEQUENCE
{
     propGrps                SEQUENCE OF PropertyGroup,
     ...
}

PropertyGroup      ::= SEQUENCE OF PropertyParm

TerminationStateDescriptor ::= SEQUENCE
{
     propertyParms           SEQUENCE OF PropertyParm,
     eventBufferControl      EventBufferControl OPTIONAL,
     serviceState            ServiceState OPTIONAL,
     ...
}

EventBufferControl ::= ENUMERATED
{
     off(0),
     lockStep(1),
     ...
}

ServiceState            ::= ENUMERATED
{
     test(0),
     outOfSvc(1),
```

```
    inSvc(2),
    ...
}

MuxDescriptor      ::= SEQUENCE
{
    muxType                MuxType,
    termList               SEQUENCE OF TerminationID,
    nonStandardData        NonStandardData OPTIONAL,
    ...
}
MuxType            ::= ENUMERATED
{
    h221(0),
    h223(1),
    h226(2),
    v76(3),
    ...,
    nx64k(4)
}

StreamID           ::= INTEGER(0..65535)   -- 16-bit unsigned integer

EventsDescriptor        ::= SEQUENCE
{
    requestID              RequestID OPTIONAL,
    -- RequestID must be present if eventList
    -- is non empty
    eventList              SEQUENCE OF RequestedEvent,
    ...
}

RequestedEvent     ::= SEQUENCE
{
    pkgdName               PkgdName,
    streamID               StreamID OPTIONAL,
    eventAction            RequestedActions OPTIONAL,
    evParList              SEQUENCE OF EventParameter,
    ...
}

RegulatedEmbeddedDescriptor ::= SEQUENCE
{
    secondEvent            SecondEventsDescriptor OPTIONAL,
    signalsDescriptor      SignalsDescriptor OPTIONAL,
    ...
}

NotifyBehaviour         ::= CHOICE
{
    notifyImmediate        NULL,
    notifyRegulated        RegulatedEmbeddedDescriptor,
    neverNotify            NULL,
    ...
}

RequestedActions        ::= SEQUENCE
{
    keepActive             BOOLEAN OPTIONAL,
    eventDM                EventDM OPTIONAL,
    secondEvent            SecondEventsDescriptor OPTIONAL,
    signalsDescriptor      SignalsDescriptor OPTIONAL,
    ...,
    notifyBehaviour        NotifyBehaviour OPTIONAL,
```

```
    resetEventsDescriptor   NULL OPTIONAL
}


EventDM             ::= CHOICE
{
    digitMapName            DigitMapName,
    digitMapValue           DigitMapValue
}

SecondEventsDescriptor  ::= SEQUENCE
{
    requestID               RequestID OPTIONAL,
    eventList               SEQUENCE OF SecondRequestedEvent,
    ...
}

SecondRequestedEvent    ::= SEQUENCE
{
    pkgdName                PkgdName,
    streamID                StreamID OPTIONAL,
    eventAction             SecondRequestedActions OPTIONAL,
    evParList               SEQUENCE OF EventParameter,
    ...
}
SecondRequestedActions  ::= SEQUENCE
{
    keepActive              BOOLEAN OPTIONAL,
    eventDM                 EventDM OPTIONAL,
    signalsDescriptor       SignalsDescriptor OPTIONAL,
    ...,
    notifyBehaviour         NotifyBehaviour OPTIONAL,
    resetEventsDescriptor   NULL OPTIONAL
}

EventBufferDescriptor   ::= SEQUENCE OF EventSpec

EventSpec           ::= SEQUENCE
{
    eventName               EventName,
    streamID                StreamID OPTIONAL,
    eventParList            SEQUENCE OF EventParameter,
    ...
}

SignalsDescriptor       ::= SEQUENCE OF SignalRequest

SignalRequest       ::=CHOICE
{
    signal                  Signal,
    seqSigList              SeqSigList,
    ...
}

SeqSigList              ::= SEQUENCE
{
    id                      INTEGER(0..65535),
    signalList              SEQUENCE OF Signal
}

Signal                  ::= SEQUENCE
{
    signalName              SignalName,
    streamID                StreamID OPTIONAL,
```

```
    sigType                 SignalType OPTIONAL,
    duration                INTEGER (0..65535) OPTIONAL,
    notifyCompletion        NotifyCompletion OPTIONAL,
    keepActive              BOOLEAN OPTIONAL,
    sigParList              SEQUENCE OF SigParameter,
    ...,
    direction               SignalDirection OPTIONAL,
    requestID               RequestID OPTIONAL,
    intersigDelay           INTEGER (0..65535) OPTIONAL
}

SignalType              ::= ENUMERATED
{
    brief(0),
    onOff(1),
    timeOut(2),
    ...
}

SignalDirection         ::= ENUMERATED
{
    internal(0),
    external(1),
    both(2),
    ...
}

SignalName              ::= PkgdName

NotifyCompletion        ::= BIT STRING
{
    onTimeOut(0), onInterruptByEvent(1),
    onInterruptByNewSignalDescr(2), otherReason(3), onIteration(4)
}

SigParameter       ::= SEQUENCE
{
    sigParameterName        Name,
    value                   Value,
    -- For use of extraInfo see the comment related to PropertyParm
    extraInfo               CHOICE
    {
        relation                Relation,
        range                   BOOLEAN,
        sublist                 BOOLEAN
    } OPTIONAL,
    ...
}

-- For an AuditCapReply with all events, the RequestID shall be ALL.
-- ALL is represented by 0xffffffff.
RequestID          ::= INTEGER(0..4294967295)   -- 32-bit unsigned integer

ModemDescriptor         ::= SEQUENCE
{
    mtl                     SEQUENCE OF ModemType,
    mpl                     SEQUENCE OF PropertyParm,
    nonStandardData         NonStandardData OPTIONAL
}

ModemType          ::= ENUMERATED
{
    v18(0),
    v22(1),
```

```
    v22bis(2),
    v32(3),
    v32bis(4),
    v34(5),
    v90(6),
    v91(7),
    synchISDN(8),
    ...
}

DigitMapDescriptor ::= SEQUENCE
{
    digitMapName            DigitMapName OPTIONAL,
    digitMapValue           DigitMapValue OPTIONAL
}

DigitMapName       ::= Name

DigitMapValue      ::= SEQUENCE
{
    startTimer              INTEGER(0..99) OPTIONAL,
    shortTimer              INTEGER(0..99) OPTIONAL,
    longTimer               INTEGER(0..99) OPTIONAL,
    digitMapBody            IA5String,
    -- Units are seconds for start, short and long timers, and hundreds
    -- of milliseconds for duration timer. Thus start, short, and long
    -- range from 1 to 99 seconds and duration from 100 ms to 9.9 s
    -- See A.3 for explanation of DigitMap syntax
    ...,
    durationTimer           INTEGER (0..99) OPTIONAL
}

ServiceChangeParm  ::= SEQUENCE
{
    serviceChangeMethod         ServiceChangeMethod,
    serviceChangeAddress        ServiceChangeAddress OPTIONAL,
    serviceChangeVersion        INTEGER(0..99) OPTIONAL,
    serviceChangeProfile        ServiceChangeProfile OPTIONAL,
    serviceChangeReason         Value,
    -- A serviceChangeReason consists of a numeric reason code
    -- and an optional text description.
    -- The serviceChangeReason shall be a string consisting of
    -- a decimal reason code, optionally followed by a single
    -- space character and a textual description string.
    -- This string is first BER-encoded as an IA5String.
    -- The result of this BER-encoding is then encoded as
    -- an ASN.1 OCTET STRING type, "double wrapping" the
    -- value as was done for package elements.
    serviceChangeDelay      INTEGER(0..4294967295) OPTIONAL,
    -- 32-bit unsigned integer
    serviceChangeMgcId          Mid OPTIONAL,
    timeStamp                   TimeNotation OPTIONAL,
    nonStandardData             NonStandardData OPTIONAL,
    ...,
    serviceChangeInfo           AuditDescriptor OPTIONAL,
    serviceChangeIncompleteFlag NULL OPTIONAL
}

ServiceChangeAddress    ::= CHOICE
{
    portNumber                  INTEGER(0..65535),   -- TCP/UDP port number
    ip4Address                  IP4Address,
    ip6Address                  IP6Address,
    domainName                  DomainName,
```

```
    deviceName                  PathName,
    mtpAddress                  OCTET STRING(SIZE(2..4)),
    ...
}

ServiceChangeResParm    ::= SEQUENCE
{
    serviceChangeMgcId          MId OPTIONAL,
    serviceChangeAddress        ServiceChangeAddress OPTIONAL,
    serviceChangeVersion        INTEGER(0..99) OPTIONAL,
    serviceChangeProfile        ServiceChangeProfile OPTIONAL,
    timestamp                   TimeNotation OPTIONAL,
    ...
}

ServiceChangeMethod     ::= ENUMERATED
{
    failover(0),
    forced(1),
    graceful(2),
    restart(3),
    disconnected(4),
    handOff(5),
    ...
}

ServiceChangeProfile    ::= SEQUENCE
{
    profileName                 IA5String(SIZE (1..67))
    -- 64 characters for name, 1 for "/", 2 for version to match ABNF
}

PackagesDescriptor ::= SEQUENCE OF PackagesItem

PackagesItem       ::= SEQUENCE
{
    packageName             Name,
    packageVersion          INTEGER(0..99),
    ...
}

StatisticsDescriptor        ::= SEQUENCE OF StatisticsParameter

StatisticsParameter         ::= SEQUENCE
{
    statName                PkgdName,
    statValue               Value OPTIONAL
}
-- If statistic consists of a sub-lists there will be more than
-- one octetstring in statValue.

NonStandardData         ::= SEQUENCE
{
    nonStandardIdentifier       NonStandardIdentifier,
    data                        OCTET STRING
}
```

```
NonStandardIdentifier   ::= CHOICE
{
     object                         OBJECT IDENTIFIER,
     h221NonStandard                H221NonStandard,
     experimental                   IA5String(SIZE(8)),
     -- first two characters should be "X-" or "X+"
     ...
}

H221NonStandard    ::= SEQUENCE
{    t35CountryCode1                INTEGER(0..255),
     t35CountryCode2                INTEGER(0..255),     -- country, as per T.35
     t35Extension                   INTEGER(0..255),     -- assigned nationally
     manufacturerCode               INTEGER(0..65535),   -- assigned nationally
     ...
}

TimeNotation       ::= SEQUENCE
{
     date                     IA5String(SIZE(8)), -- yyyymmdd format
     time                     IA5String(SIZE(8))  -- hhmmssss format
     -- per ISO 8601:2004
}

Value ::= SEQUENCE OF OCTET STRING

END
```

A.3 DigitMaps and path names

From a syntactic viewpoint, DigitMaps are strings with syntactic restrictions imposed upon them. The syntax of valid DigitMaps is specified in ABNF (RFC 2234). The syntax for DigitMaps presented in this clause is for illustrative purposes only. The definition of `digitMap` in Annex B takes precedence in the case of differences between the two.

```
digitMap                = (digitString / LWSP "(" LWSP digitStringList LWSP
                          ")" LWSP)

digitStringList         = digitString *(LWSP "|" LWSP digitString)

digitString             = 1*(digitStringElement)

digitStringElement      = digitPosition [DOT]

digitPosition           = digitMapLetter / digitMapRange

digitMapRange           = ("x" / (LWSP "[" LWSP digitLetter LWSP "]" LWSP))

digitLetter             = *((DIGIT "-" DIGIT) / digitMapLetter)

digitMapLetter          = DIGIT                ;Basic event symbols
                          / %x41-4B / %x61-6B  ; a-k, A-K
                          / "L" / "S" / "T"    ;Inter-event timers
                                               ; (long, short, start)
                          / "Z"                ;Long duration modifier

DOT                     = %x2E                 ; "."

SP                      = %x20                 ; space

HTAB                    = %x09                 ; horizontal tab

CR                      = %x0D                 ; Carriage return
```

```
LF                      = %x0A                  ; linefeed

LWSP                    = *(WSP / COMMENT / EOL)

EOL                     = (CR [LF] / LF)

WSP                     = SP / HTAB             ; white space

SafeChar                = DIGIT / ALPHA / "+" / "-" / "&" /
                          "!" / "_" / "/" / "'" / "?" / "@" /
                          "^" / "`" / "~" / "*" / "$" / "\" /
                          "(" / ")" / "%" / "|" / "."

RestChar                = ";" / "[" / "]" / "{" / "}" / ":" / "," / "#" /
                          "<" / ">" / "="

DIGIT                   = %x30-39               ; 0-9

ALPHA                   = %x41-5A / %x61-7A     ; A-Z / a-z
```

A path name is also a string with syntactic restrictions imposed upon it. The ABNF production defining it is copied from Annex B.

```
; Total length of pathNAME must not exceed 64 chars.

pathname        = ["*"] NAME *("/" / "*"/ ALPHA / DIGIT /"_" / "$" )
                  ["@" pathDomainName ]

; ABNF allows two or more consecutive "." although it is meaningless

; in a path domain name.

pathDomainName = (ALPHA / DIGIT / "*" ) *63(ALPHA / DIGIT / "-" / "*" / ".")

NAME            = ALPHA *63(ALPHA / DIGIT / "_")
```

Annex B

Text encoding of the protocol

B.1 Coding of wildcards

In a text encoding of the protocol, while TerminationIDs are arbitrary, by judicious choice of names, the wildcard character, "*" may be made more useful. When the wildcard character is encountered, it will "match" all TerminationIDs having the same previous and following characters (if appropriate). For example, if there were TerminationIDs of R13/3/1, R13/3/2 and R13/3/3, the TerminationID R13/3/* would match all of them. There are some circumstances where all terminations must be referred to. The TerminationID "*" suffices, and is referred to as ALL. The CHOOSE TerminationID "$" may be used to signal to the MG that it has to create an ephemeral termination or select an idle physical termination.

B.2 ABNF specification

The protocol syntax is presented in ABNF according to RFC 2234.

NOTE 1 – This syntax specification does not enforce all restrictions on element inclusions and values. Some additional restrictions are stated in comments and other restrictions appear in the text of this Recommendation. These additional restrictions are part of the protocol even though not enforced by this Recommendation.

NOTE 2 – The syntax is context-dependent. For example, "Add" can be the AddToken or a NAME depending on the context in which it occurs.

Everything in the ABNF and text encoding is case insensitive. This includes TerminationIDs, DigitMapIDs, etc. SDP is case sensitive as per RFC 2327.

```
; NOTE - The ABNF in this section uses the VALUE construct (or lists of
; VALUE constructs) to encode various package element values (properties,
; signal parameters, etc.).  The types of these values vary and are
; specified in the relevant package definition.  Several such types are
; described in 12.2.
;
; The ABNF specification for VALUE allows a quotedString form or a
; collection of SafeChars.  The encoding of package element values into
; ABNF VALUES is specified below.  If a type's encoding allows characters
; other than SafeChars, the quotedString form must be used for all values
; of that type, even for specific values that consist only of SafeChars.
;
; String:  A string must use the quotedString form of VALUE and can
; contain anything allowable in the quotedString form.
;
; Integer, Double, and Unsigned Integer:  Decimal values can be encoded
; using characters 0-9.  Hexadecimal values must be prefixed with '0x'
; and can use characters 0-9,a-f,A-F.  An octal format is not supported.
; Negative integers start with '-' and must be Decimal.  The SafeChar
; form of VALUE must be used.
;
; Character:  A UTF-8 encoding of a single letter surrounded by double
; quotes.
;
; Enumeration:  An enumeration must use the SafeChar form of VALUE
; and can contain anything allowable in the SafeChar form.
;
; Boolean:  Boolean values are encoded as "on" and "off" and are
; case insensitive.  The SafeChar form of VALUE must be used.
;
; Future types:  Any defined types must fit within
; the ABNF specification of VALUE.  Specifically, if a type's encoding
; allows characters other than SafeChars, the quotedString form must
; be used for all values of that type, even for specific values that
; consist only of SafeChars.
;
; Note that there is no way to use the double quote character within
; a value.
;
; Note that SDP disallows whitespace at the beginning of a line, Megaco
; ABNF allows whitespace before the beginning of the SDP in the
; Local/Remote descriptor.  Parsers should accept whitespace between the
; LBRKT following the Local/Remote token and the beginning of the SDP.

megacoMessage           = LWSP [authenticationHeader SEP] message

authenticationHeader    = AuthToken EQUAL SecurityParmIndex COLON
                          SequenceNum COLON AuthData

SecurityParmIndex       = "0x" 8(HEXDIG)
SequenceNum             = "0x" 8(HEXDIG)
AuthData                = "0x" 24*64(HEXDIG)

Message                 = MegacopToken SLASH Version SEP mId SEP messageBody
; The version of the protocol defined here is equal to 3.

messageBody             = (errorDescriptor / transactionList)
```

```
transactionList          = 1*(transactionRequest / transactionReply /
                           transactionPending / transactionResponseAck /
                           segmentReply)
;Use of response acks is dependent on underlying transport

transactionPending       = PendingToken EQUAL TransactionID LBRKT RBRKT

transactionResponseAck   = ResponseAckToken LBRKT transactionAck
                           *(COMMA transactionAck) RBRKT
transactionAck           = TransactionID / (TransactionID "-" TransactionID)

transactionRequest       = TransToken EQUAL TransactionID LBRKT
                          actionRequest *(COMMA actionRequest) RBRKT

actionRequest            = CtxToken EQUAL ContextID LBRKT ((contextRequest
                           [COMMA  commandRequestList]) /
                           commandRequestList) RBRKT

contextRequest           = ((contextProperties [COMMA contextAudit])
                           / contextAudit)

contextProperties        = contextProperty *(COMMA contextProperty)

; at-most-once
; EmergencyOff to be used in MG to MGC direction only in H.248.1 V1 and V2
; EmergencyToken or EmergencyOffToken, but not both
contextProperty          = (topologyDescriptor / priority / EmergencyToken /
                           EmergencyOffToken / iepsValue /
                           contextAttrDescriptor)

contextAudit             = ContextAuditToken LBRKT (contextAuditProperties
                           *(COMMA contextAuditProperties)) /
                             indAudcontextAttrDescriptor     RBRKT

; at-most-once except contextAuditSelector
contextAuditProperties   = (TopologyToken / EmergencyToken /
                           PriorityToken /IEPSToken/ pkgdName /
                           contextAuditSelector)


; at-most-once
contextAuditSelector     = priority / emergencyValue / iepsValue /
                           contextAttrDescriptor / auditSelectLogic

auditSelectLogic         = [ AndAUDITselectToken / OrAUDITselectToken ]
; If empty, AND of selection conditions is assumed.

indAudcontextAttrDescriptor
                         = ContextAttrToken LBRKT contextAuditProperties
                           *(COMMA contextAuditProperties) RBRKT

; "O-" indicates an optional command
; "W-" indicates a wildcarded response to a command
commandRequestList       = ["O-"] ["W-"] commandRequest *(COMMA ["O-"]
                           ["W-"]commandRequest)

commandRequest           = (ammRequest / subtractRequest / auditRequest /
                           notifyRequest / serviceChangeRequest)

transactionReply         = ReplyToken EQUAL TransactionID [SLASH segmentNumber
                           [SLASH SegmentationCompleteToken]] LBRKT
                           [ImmAckRequiredToken COMMA]
                           (errorDescriptor / actionReplyList) RBRKT
```

```
segmentReply            = MessageSegmentToken EQUAL TransactionID SLASH
                          segmentNumber [SLASH SegmentationCompleteToken]

segmentNumber           = UINT16

actionReplyList         = actionReply *(COMMA actionReply)

actionReply             = CtxToken EQUAL ContextID [LBRKT (errorDescriptor /
                          commandReply /(commandReply COMMA
                          errorDescriptor)) RBRKT]

commandReply            = ((contextProperties [COMMA commandReplyList]) /
                          commandReplyList)

commandReplyList        = commandReplys *(COMMA commandReplys)

commandReplys           = (serviceChangeReply / auditReply / ammsReply /
                          notifyReply)

;Add, Move and Modify have the same request parameters
ammRequest              = (AddToken / MoveToken / ModifyToken) EQUAL
                          termIDList [LBRKT ammParameter *(COMMA
                          ammParameter) RBRKT]

;at-most-once
ammParameter            = (mediaDescriptor / modemDescriptor /muxDescriptor /
                          eventsDescriptor / signalsDescriptor /
                          digitMapDescriptor / eventBufferDescriptor /
                          auditDescriptor / statisticsDescriptor)

ammsReply               = (AddToken / MoveToken / ModifyToken / SubtractToken)
                          EQUAL termIDList [LBRKT terminationAudit RBRKT]

subtractRequest         = SubtractToken EQUAL termIDList [LBRKT
                          auditDescriptor RBRKT]

auditRequest            = (AuditValueToken / AuditCapToken) EQUAL
                          termIDList LBRKT auditDescriptor RBRKT

auditReply              = (AuditValueToken / AuditCapToken)
                          (contextTerminationAudit / auditOther)

auditOther              = EQUAL termIDList [LBRKT terminationAudit RBRKT]

terminationAudit        = auditReturnParameter *(COMMA auditReturnParameter)

contextTerminationAudit = EQUAL CtxToken (terminationIDList / LBRKT
                          errorDescriptor RBRKT)

auditReturnParameter    = (mediaDescriptor / modemDescriptor / muxDescriptor /
                          eventsDescriptor / signalsDescriptor /
                          digitMapDescriptor / observedEventsDescriptor /
                          eventBufferDescriptor / statisticsDescriptor /
                          packagesDescriptor / errorDescriptor /
                          auditReturnItem)

auditReturnItem         = (MuxToken / ModemToken / MediaToken /
                          DigitMapToken / StatsToken / ObservedEventsToken /
                          PackagesToken)

auditDescriptor         = AuditToken LBRKT [auditItem *(COMMA auditItem)]
                          RBRKT
```

```
notifyRequest           = NotifyToken EQUAL termIDList LBRKT
                          (observedEventsDescriptor [COMMA errorDescriptor])
                          RBRKT

notifyReply             = NotifyToken EQUAL termIDList [LBRKT errorDescriptor
                          RBRKT]

serviceChangeRequest    = ServiceChangeToken EQUAL termIDList LBRKT
                          serviceChangeDescriptor RBRKT

serviceChangeReply      = ServiceChangeToken EQUAL termIDList [LBRKT
                          (errorDescriptor / serviceChangeReplyDescriptor)
                          RBRKT]

errorDescriptor         = ErrorToken EQUAL ErrorCode LBRKT [quotedString]
                          RBRKT

ErrorCode               = 1*4(DIGIT) ; could be extended

TransactionID           = UINT32

mId                     = ((domainAddress / domainName) [":" portNumber]) /
                          mtpAddress / deviceName

; ABNF allows two or more consecutive "." although it is meaningless
; in a domain name.
domainName              = "<" (ALPHA / DIGIT) *63(ALPHA / DIGIT / "-" / ".")
                          ">"

deviceName              = pathNAME

;The values 0x0, 0xFFFFFFFE and 0xFFFFFFFF are reserved.
;'-' is used for NULL context. '*' is ALL. '$' is CHOOSE.
ContextID               = (UINT32 / "*" / "-" / "$")

domainAddress           = "[" (IPv4address / IPv6address) "]"

;RFC 2373 contains the definition of IPv6 addresses.
IPv6address             = hexpart [":" IPv4address]

IPv4address             = V4hex DOT V4hex DOT V4hex DOT V4hex

V4hex                   = 1*3(DIGIT) ; "0".."255"

; this production, while occurring in RFC 2373, is not referenced
; IPv6prefix            = hexpart SLASH 1*2DIGIT

hexpart                 = hexseq "::" [hexseq] / "::" [hexseq] / hexseq

hexseq                  = hex4 *(":" hex4)

hex4                    = 1*4HEXDIG

portNumber              = UINT16

; Addressing structure of mtpAddress:
; 25 - 15           0
;    |  PC        | NI |
;    24 - 14 bits    2 bits
; NOTE – 14 bits are defined for international use.
; Two national options exist where the point code is 16 or 24 bits.
; To octet align the mtpAddress the MSBs shall be encoded as 0s.
; An octet shall be represented by 2 hex digits.
mtpAddress              = MTPToken LBRKT 4*8 (HEXDIG) RBRKT
```

```
termIDList              = (TerminationID / LSBRKT TerminationID 1*(COMMA
                          TerminationID) RSBRKT)

terminationIDList       = LBRKT TerminationID *(COMMA TerminationID) RBRKT

; Total length of pathNAME must not exceed 64 chars.
pathNAME                = ["*"] NAME *("/" / "*"/ ALPHA / DIGIT /"_" / "$")
                          ["@" pathDomainName]

; ABNF allows two or more consecutive "." although it is meaningless
; in a path domain name.
pathDomainName          = (ALPHA / DIGIT / "*") *63(ALPHA / DIGIT / "-" /
                          "*" / ".")

; '*' is ALL. '$' is CHOOSE.
TerminationID           = "ROOT" / pathNAME / "$" / "*"

mediaDescriptor         = MediaToken LBRKT mediaParm *(COMMA mediaParm) RBRKT

; at-most one terminationStateDescriptor
; and either streamParm(s) or streamDescriptor(s) but not both
mediaParm               = (streamParm / streamDescriptor /
                          terminationStateDescriptor)

; at-most-once per item
streamParm              = (localDescriptor / remoteDescriptor /
                          localControlDescriptor / statisticsDescriptor)

streamDescriptor        = StreamToken EQUAL StreamID LBRKT streamParm *(COMMA
                          streamParm) RBRKT

localControlDescriptor  = LocalControlToken LBRKT localParm *(COMMA localParm)
                          RBRKT

; at-most-once per item except for propertyParm
localParm               = (streamMode / propertyParm / reservedValueMode /
                          reservedGroupMode)

reservedValueMode       = ReservedValueToken EQUAL ("ON" / "OFF")

reservedGroupMode       = ReservedGroupToken EQUAL ("ON" / "OFF")

streamMode              = ModeToken EQUAL streamModes

streamModes             = (SendonlyToken / RecvonlyToken / SendrecvToken /
                          InactiveToken / LoopbackToken)

propertyParm            = pkgdName parmValue

; the (Safe)Char '$' means CHOOSE
; the (Safe)Char '*' means ALL
parmValue               = (EQUAL alternativeValue / INEQUAL VALUE)

alternativeValue        = (VALUE
                          / LSBRKT VALUE *(COMMA VALUE) RSBRKT
                              ; sublist (i.e., A AND B AND ...)
                          / LBRKT VALUE *(COMMA VALUE) RBRKT
                              ; alternatives (i.e., A OR B OR ...)
                          / LSBRKT VALUE COLON VALUE RSBRKT)
                              ; range

INEQUAL                 = LWSP (">" / "<" / "#") LWSP ; '#' means "not equal"
```

```
LSBRKT                    = LWSP "[" LWSP

RSBRKT                    = LWSP "]" LWSP

; NOTE - The octet zero is not among the permitted characters in octet
; string. As the current definition is limited to SDP, and a zero octet
; would not be a legal character in SDP, this is not a concern.
localDescriptor           = LocalToken LBRKT octetString RBRKT

remoteDescriptor          = RemoteToken LBRKT octetString RBRKT

eventBufferDescriptor     = EventBufferToken [LBRKT eventSpec*(COMMA eventSpec)
                            RBRKT]

eventSpec                 = pkgdName [LBRKT eventSpecParameter *(COMMA
                            eventSpecParameter) RBRKT]

eventSpecParameter        = (eventStream / eventOther)

eventBufferControl        = BufferToken EQUAL eventBufferControlValue
eventBufferControlValue   = ("OFF" / LockStepToken)

terminationStateDescriptor
                          = TerminationStateToken LBRKT terminationStateParm
                            *(COMMA terminationStateParm) RBRKT

; at-most-once per item except for propertyParm
terminationStateParm      = (propertyParm / serviceStates / eventBufferControl)

serviceStates             = ServiceStatesToken EQUAL serviceStatesValue
serviceStatesValue        = (TestToken / OutOfSvcToken / InSvcToken)

muxDescriptor             = MuxToken EQUAL MuxType terminationIDList

MuxType                   = (H221Token / H223Token / H226Token / V76Token /
                            extensionParameter / Nx64kToken)

StreamID                  = UINT16

pkgdName                  = (PackageName SLASH ItemID);specific item
                            / (PackageName SLASH "*") ;all items in package
                            / ("*" SLASH "*")    ; all items supported by the MG

PackageName               = NAME

ItemID                    = NAME

eventsDescriptor          = EventsToken [EQUAL RequestID LBRKT requestedEvent
                            *(COMMA requestedEvent) RBRKT]

requestedEvent            = pkgdName [LBRKT eventParameter *(COMMA
                            eventParameter) RBRKT]

notifyRegulated           = NotifyRegulatedToken [LBRKT (embedWithSig/
                            embedNoSig) RBRKT]

notifyBehaviour           = NotifyImmediateToken / notifyRegulated /
                            NeverNotifyToken

; at-most-once each of KeepActiveToken, notifyBehaviour,eventDM,
; ResetEventsDescriptor and eventStream
; at most one of either embedWithSig or embedNoSig but not both
; KeepActiveToken and embedWithSig must not both be present
eventParameter            = (embedWithSig / embedNoSig / KeepActiveToken /
```

```
                          eventDM / eventStream / eventOther /
                          notifyBehaviour / ResetEventsDescriptorToken)

embedWithSig            = EmbedToken LBRKT signalsDescriptor [COMMA
                          embedFirst] RBRKT

embedNoSig              = EmbedToken LBRKT embedFirst RBRKT

; at-most-once of each
embedFirst              = EventsToken [EQUAL RequestID LBRKT
                          secondRequestedEvent *(COMMA secondRequestedEvent)
                          RBRKT]

secondRequestedEvent    = pkgdName [LBRKT secondEventParameter *(COMMA
                          secondEventParameter) RBRKT]

; at-most-once each of embedSig, KeepActiveToken, notifyBehaviour,eventDM
; ResetEventsDescriptor or eventStream
; KeepActiveToken and embedSig must not both be present
secondEventParameter    = (embedSig / KeepActiveToken / eventDM /
                          eventStream / eventOther / notifyBehaviour /
                          ResetEventsDescriptorToken)

embedSig                = EmbedToken LBRKT signalsDescriptor RBRKT

eventStream             = StreamToken EQUAL StreamID

eventOther              = eventParameterName parmValue

eventParameterName      = NAME

eventDM                 = DigitMapToken EQUAL((digitMapName) / (LBRKT
                          digitMapValue RBRKT))

signalsDescriptor       = SignalsToken [LBRKT signalParm *(COMMA signalParm)
                          RBRKT]

signalParm              = signalList / signalRequest

signalRequest           = signalName [LBRKT sigParameter *(COMMA sigParameter)
                          RBRKT]

signalList              = SignalListToken EQUAL signalListId LBRKT
                          signalListParm *(COMMA signalListParm) RBRKT

signalListId            = UINT16

;exactly once signalType, at most once duration and every signal
;parameter
signalListParm          = signalRequest

signalName              = pkgdName

;at-most-once sigStream, at-most-once sigSignalType,
;at-most-once sigDuration, at-most-once sigDirection,
;at-most-once sigRequestID, at-most-once sigIntsigDelay
;every signalParameterName at most once
sigParameter            = sigStream / sigSignalType / sigDuration / sigOther /
                          notifyCompletion / KeepActiveToken /
                          sigDirection / sigRequestID / sigIntsigDelay

sigStream               = StreamToken EQUAL StreamID

sigOther                = sigParameterName parmValue
```

```
sigParameterName        = NAME

sigSignalType           = SignalTypeToken EQUAL signalType

signalType              = (OnOffToken / TimeOutToken / BriefToken)

sigDuration             = DurationToken EQUAL UINT16

sigDirection            = DirectionToken EQUAL direction

direction               = ExternalToken / InternalToken / BothToken

sigRequestID            = RequestIDToken EQUAL RequestID

sigIntsigDelay          = IntsigDelayToken EQUAL UINT16

notifyCompletion        = NotifyCompletionToken EQUAL (LBRKT
                          notificationReason *(COMMA notificationReason)
                          RBRKT)

notificationReason      = TimeOutToken / InterruptByEventToken /
                          InterruptByNewSignalsDescrToken / OtherReasonToken /
                          IterationToken

observedEventsDescriptor
                        = ObservedEventsToken EQUAL RequestID LBRKT
                          observedEvent *(COMMA observedEvent) RBRKT

; time per event, because it might be buffered
observedEvent           = [TimeStamp LWSP COLON] LWSP pkgdName [LBRKT
                          observedEventParameter *(COMMA
                          observedEventParameter) RBRKT]

; at-most-once eventStream, every eventParameterName at most once
observedEventParameter  = eventStream / eventOther

; For an AuditCapReply with all events, the RequestID should be ALL.
RequestID               = UINT32 / "*"

modemDescriptor         = ModemToken ((EQUAL modemType) / (LSBRKT modemType
                          *(COMMA modemType) RSBRKT)) [LBRKT propertyParm
                          *(COMMA propertyParm) RBRKT]

; at-most-once except for extensionParameter
modemType               = V32bisToken / V22bisToken / V18Token / V22Token /
                          V32Token / V34Token / V90Token / V91Token /
                          SynchISDNToken / extensionParameter)

digitMapDescriptor      = DigitMapToken EQUAL ((LBRKT digitMapValue RBRKT) /
                          (digitMapName [LBRKT digitMapValue RBRKT]))

digitMapName            = NAME

digitMapValue           = ["T" COLON Timer COMMA] ["S" COLON Timer COMMA] ["L"
                          COLON Timer COMMA] ["Z" COLON Timer COMMA] digitMap

Timer                   = 1*2DIGIT
; Units are seconds for T, S, and L timers, and hundreds of
; milliseconds for Z timer. Thus T, S, and L range from 1 to 99
; seconds and Z from 100 ms to 9.9 s

digitMap                = (digitString / LWSP "(" LWSP digitStringList LWSP
                          ")" LWSP)
```

```
digitStringList         = digitString *(LWSP "|" LWSP digitString)

digitString             = 1*(digitStringElement)

digitStringElement      = digitPosition [DOT]

digitPosition           = digitMapLetter / digitMapRange

digitMapRange           = ("x" / (LWSP "[" LWSP digitLetter LWSP "]" LWSP))

digitLetter             = *((DIGIT "-" DIGIT) / digitMapLetter)

digitMapLetter          = DIGIT               ;Basic event symbols
                          / %x41-4B / %x61-6B ; a-k, A-K
                          / "L" / "S" / "T"   ;Inter-event timers
                                              ; (long, short, start)
                          / "Z"               ;Long duration modifier

; at-most-once, and DigitMapToken and PackagesToken are not allowed
; in AuditCapabilities command
auditItem               = auditReturnItem / SignalsToken / EventBufferToken /
                          EventsToken / indAudterminationAudit

indAudterminationAudit  = indAudauditReturnParameter *(COMMA
                          indAudauditReturnParameter)

indAudauditReturnParameter
                        = indAudmediaDescriptor / indAudeventsDescriptor /
                          indAudsignalsDescriptor /
                          indAuddigitMapDescriptor /
                          indAudeventBufferDescriptor /
                          indAudstatisticsDescriptor /
                          indAudpackagesDescriptor

indAudmediaDescriptor   = MediaToken LBRKT indAudmediaParm *(COMMA
                          indAudmediaParm) RBRKT

; either streamParm or streamDescriptor but not both
indAudmediaParm         = indAudstreamParm / indAudstreamDescriptor /
                          indAudterminationStateDescriptor

; at-most-once
indAudstreamParm        = (indAudlocalControlDescriptor /
                          indAudstatisticsDescriptor    /
                          indAudremoteDescriptor / indAudlocalDescriptor )

indAudremoteDescriptor  = RemoteToken LBRKT octetString RBRKT
indAudlocalDescriptor   = LocalToken LBRKT octetString RBRKT

indAudstreamDescriptor  = StreamToken EQUAL StreamID LBRKT indAudstreamParm
                          RBRKT

indAudlocalControlDescriptor
                        = LocalControlToken LBRKT indAudlocalParm
                          *(COMMA indAudlocalParm) RBRKT

; at-most-once per item
indAudlocalParm         = ModeToken [(EQUAL/INEQUAL) streamModes]/ pkgdName /
                          propertyParm / ReservedValueToken /
                          ReservedGroupToken
; propertyparm and streamModes are used only to specify audit selection
; criteria. AND/OR selection logic is specified at context level.
```

```
indAudterminationStateDescriptor
                        = TerminationStateToken LBRKT
                          indAudterminationStateParm RBRKT

; at-most-once per item
indAudterminationStateParm
                        = pkgdName / propertyParm / ServiceStatesToken
                          [(EQUAL/INEQUAL) serviceStatesValue ] / BufferToken
; When values are included a Select operation is implied.
; AND/OR logic is specified at context level.

indAudeventBufferDescriptor
                        = EventBufferToken LBRKT indAudeventSpec RBRKT

indAudeventSpec         = pkgdName [LBRKT indAudeventSpecParameter RBRKT]

indAudeventSpecParameter= eventStream / eventParameterName

indAudeventsDescriptor  = EventsToken [EQUAL RequestID] LBRKT
                          indAudrequestedEvent RBRKT

indAudrequestedEvent    = pkgdName

indAudsignalsDescriptor = SignalsToken LBRKT [indAudsignalParm] RBRKT

indAudsignalParm        = indAudsignalList / indAudsignalRequest

indAudsignalRequest     = signalName [LBRKT indAudsignalRequestParm
                          *(COMMA indAudsignalRequestParm) RBRKT]

indAudsignalRequestParm = sigStream / sigRequestID

indAudsignalList        = SignalListToken EQUAL signalListId [LBRKT
                          indAudsignalListParm RBRKT]

indAudsignalListParm    = indAudsignalRequest

indAuddigitMapDescriptor= DigitMapToken EQUAL (digitMapName)

indAudstatisticsDescriptor
                        = StatsToken LBRKT pkgdName RBRKT

indAudpackagesDescriptor= PackagesToken LBRKT packagesItem RBRKT

serviceChangeDescriptor = ServicesToken LBRKT serviceChangeParm *(COMMA
                          serviceChangeParm) RBRKT

; each parameter at-most-once, except auditItem
; at most one of either serviceChangeAddress or serviceChangeMgcId but
; not both
; serviceChangeMethod and serviceChangeReason are REQUIRED
serviceChangeParm       = serviceChangeMethod / serviceChangeReason /
                          serviceChangeDelay / serviceChangeAddress /
                          serviceChangeProfile / extension / TimeStamp /
                          serviceChangeMgcId / serviceChangeVersion /
                          ServiceChangeIncompleteToken / auditItem)

serviceChangeReplyDescriptor
                        = ServicesToken LBRKT servChgReplyParm *(COMMA
                          servChgReplyParm) RBRKT

; at-most-once. Version is REQUIRED on first ServiceChange response
; at most one of either serviceChangeAddress or serviceChangeMgcId but
; not both
```

```
servChgReplyParm        = serviceChangeAddress / serviceChangeMgcId /
                          serviceChangeProfile / serviceChangeVersion /
                          TimeStamp

serviceChangeMethod     = MethodToken EQUAL (FailoverToken / ForcedToken /
                          GracefulToken / RestartToken / DisconnectedToken /
                          HandOffToken / extensionParameter)

; A serviceChangeReason consists of a numeric reason code
; and an optional text description.
; A serviceChangeReason must be encoded using the quotedString
; form of VALUE.
; The quotedString shall contain a decimal reason code,
; optionally followed by a single space character and a
; textual description string.
serviceChangeReason     = ReasonToken EQUAL VALUE

serviceChangeDelay      = DelayToken EQUAL UINT32

serviceChangeAddress    = ServiceChangeAddressToken EQUAL (mId / portNumber)

serviceChangeMgcId      = MgcIdToken EQUAL mId

serviceChangeProfile    = ProfileToken EQUAL NAME SLASH Version

serviceChangeVersion    = VersionToken EQUAL Version

extension               = extensionParameter parmValue

packagesDescriptor      = PackagesToken LBRKT packagesItem *(COMMA
                          packagesItem) RBRKT

Version                 = 1*2(DIGIT)

packagesItem            = NAME "-" UINT16

TimeStamp               = Date "T" Time ; per ISO 8601:2004

; Date = yyyymmdd
Date                    = 8(DIGIT)

; Time = hhmmssss
Time                    = 8(DIGIT)

statisticsDescriptor    = StatsToken LBRKT statisticsParameter *(COMMA
                          statisticsParameter) RBRKT

;at-most-once per item
statisticsParameter     = pkgdName [EQUAL VALUE /
                          (LSBRKT VALUE *(COMMA VALUE) RSBRKT)]

topologyDescriptor      = TopologyToken LBRKT topologyTriple *(COMMA
                          topologyTriple) RBRKT

topologyTriple          = terminationA COMMA terminationB COMMA
                          topologyDirection [COMMA eventStream]

terminationA            = TerminationID

terminationB            = TerminationID

topologyDirection       = BothwayToken / IsolateToken / OnewayToken /
                          OnewayExternalToken / OnewayBothToken
```

```
priority                = PriorityToken EQUAL UINT16

iepsValue               = IEPSToken EQUAL ("ON" / "OFF")

emergencyValue          = EmergencyValueToken EQUAL (EmergencyToken /
                          EmergencyOffToken)

contextAttrDescriptor   = ContextAttrToken LBRKT (contextIdList /
                          propertyParm *(COMMA propertyParm)) RBRKT

; When using the contextIdList construct, the ContextID in the
; actionReply construct shall be the same as the ContextID in the
; associated actionRequest
contextIdList           = ContextListToken EQUAL LBRKT ContextID
                          *(COMMA ContextID) RBRKT

extensionParameter      = "X" ("-" / "+") 1*6(ALPHA / DIGIT)

; octetString is used to describe SDP defined in RFC 2327.
; Caution should be taken if CRLF in RFC 2327 is used.
; To be safe, use EOL in this ABNF.
; Whenever "}" appears in SDP, it is escaped by "\", e.g., "\}"
octetString             = *(nonEscapeChar)

nonEscapeChar           = ("\}" / %x01-7C / %x7E-FF)

; NOTE - The double-quote character is not allowed in quotedString.
quotedString            = DQUOTE *(SafeChar / EOL / %x80-FF / RestChar / WSP)
                          DQUOTE

UINT16                  = 1*5(DIGIT) ; %x0-FFFF

UINT32                  = 1*10(DIGIT) ; %x0-FFFFFFFF

NAME                    = ALPHA *63(ALPHA / DIGIT / "_")

VALUE                   = quotedString / 1*(SafeChar / %x80-FF)

SafeChar                = DIGIT / ALPHA / "+" / "-" / "&" /
                          "!" / "_" / "/" / "'" / "?" / "@" /
                          "^" / "`" / "~" / "*" / "$" / "\" /
                          "(" / ")" / "%" / "|" / "."

EQUAL                   = LWSP %x3D LWSP      ; "="

COLON                   = %x3A                ; ":"

LBRKT                   = LWSP %x7B LWSP      ; "{"

RBRKT                   = LWSP %x7D LWSP      ; "}"

COMMA                   = LWSP %x2C LWSP      ; ","

DOT                     = %x2E                ; "."

SLASH                   = %x2F                ; "/"

ALPHA                   = %x41-5A / %x61-7A   ; A-Z / a-z

DIGIT                   = %x30-39             ; 0-9

DQUOTE                  = %x22                ; " (Double Quote)

HEXDIG                  = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

```
SP                      = %x20                  ; space

HTAB                    = %x09                  ; horizontal tab

CR                      = %x0D                  ; Carriage return

LF                      = %x0A                  ; linefeed

LWSP                    = *(WSP / COMMENT / EOL)

EOL                     = (CR [LF] / LF)

WSP                     = SP / HTAB             ; white space

SEP                     = (WSP / EOL / COMMENT) LWSP

COMMENT                 = ";" *(SafeChar/ RestChar / WSP / %x22) EOL

RestChar                = ";" / "[" / "]" / "{" / "}" / ":" / "," / "#" /
                          "<" / ">" / "="

; New Tokens added to sigParameter must take the format of SPA*
; * may be of any form i.e., SPAM
; New Tokens added to eventParameter must take the form of EPA*
; * may be of any form i.e., EPAD

AddToken                = ("Add"                    / "A")
AndAUDITSelectToken     = ("ANDLgc")
AuditToken              = ("Audit"                  / "AT")
AuditCapToken           = ("AuditCapability"        / "AC")
AuditValueToken         = ("AuditValue"             / "AV")
AuthToken               = ("Authentication"         / "AU")
BothToken               = ("Both"                   / "B")
BothwayToken            = ("Bothway"                / "BW")
BriefToken              = ("Brief"                  / "BR")
BufferToken             = ("Buffer"                 / "BF")
CtxToken                = ("Context"                / "C")
ContextAuditToken       = ("ContextAudit"           / "CA")
ContextAttrToken        = ("ContextAttr"            / "CT")
ContextListToken        = ("ContextList"            / "CLT")
DigitMapToken           = ("DigitMap"               / "DM")
DirectionToken          = ("SPADirection"           / "SPADI")
DisconnectedToken       = ("Disconnected"           / "DC")
DelayToken              = ("Delay"                  / "DL")
DurationToken           = ("Duration"               / "DR")
EmbedToken              = ("Embed"                  / "EM")
EmergencyToken          = ("Emergency"              / "EG")
EmergencyOffToken       = ("EmergencyOff"           / "EGO")
EmergencyValueToken     = ("EmergencyValue"         / "EGV")
ErrorToken              = ("Error"                  / "ER")
EventBufferToken        = ("EventBuffer"            / "EB")
EventsToken             = ("Events"                 / "E")
ExternalToken           = ("External"               / "EX")
FailoverToken           = ("Failover"               / "FL")
ForcedToken             = ("Forced"                 / "FO")
GracefulToken           = ("Graceful"               / "GR")
H221Token               = ("H221")
H223Token               = ("H223")
H226Token               = ("H226")
HandOffToken            = ("HandOff"                / "HO")
IEPSToken               = ("IEPSCall"               / "IEPS")
ImmAckRequiredToken     = ("ImmAckRequired"         / "IA")
InactiveToken           = ("Inactive"               / "IN")
```

```
InternalToken           = ("Internal"               / "IT")
IntsigDelayToken        = ("Intersignal"            / "SPAIS")
IsolateToken            = ("Isolate"                / "IS")
InSvcToken              = ("InService"              / "IV")
InterruptByEventToken   = ("IntByEvent"             / "IBE")
InterruptByNewSignalsDescrToken
                        = ("IntBySigDescr"          / "IBS")
IterationToken          = ("Iteration"              / "IR")
KeepActiveToken         = ("KeepActive"             / "KA")
LocalToken              = ("Local"                  / "L")
LocalControlToken       = ("LocalControl"           / "O")
LockStepToken           = ("LockStep"               / "SP")
LoopbackToken           = ("Loopback"               / "LB")
MediaToken              = ("Media"                  / "M")
MegacopToken            = ("MEGACO"                 / "!")
MessageSegmentToken     = ("Segment"                / "SM")
MethodToken             = ("Method"                 / "MT")
MgcIdToken              = ("MgcIdToTry"             / "MG")
ModeToken               = ("Mode"                   / "MO")
ModifyToken             = ("Modify"                 / "MF")
ModemToken              = ("Modem"                  / "MD")
MoveToken               = ("Move"                   / "MV")
MTPToken                = ("MTP")
MuxToken                = ("Mux"                    / "MX")
NeverNotifyToken        = ("NeverNotify"            / "NBNN")
NotifyToken             = ("Notify"                 / "N")
NotifyCompletionToken   = ("NotifyCompletion"       / "NC")
NotifyImmediateToken    = ("ImmediateNotify"        / "NBIN")
NotifyRegulatedToken    = ("RegulatedNotify"        / "NBRN")
Nx64kToken              = ("Nx64Kservice"           / "N64")
ObservedEventsToken     = ("ObservedEvents"         / "OE")
OnewayToken             = ("Oneway"                 / "OW")
OnewayBothToken         = ("OnewayBoth"             / "OWB")
OnewayExternalToken     = ("OnewayExternal"         / "OWE")
OnOffToken              = ("OnOff"                  / "OO")
OrAUDITselectToken      = ("ORLgc")
OtherReasonToken        = ("OtherReason"            / "OR")
OutOfSvcToken           = ("OutOfService"           / "OS")
PackagesToken           = ("Packages"               / "PG")
PendingToken            = ("Pending"                / "PN")
PriorityToken           = ("Priority"               / "PR")
ProfileToken            = ("Profile"                / "PF")
ReasonToken             = ("Reason"                 / "RE")
RecvonlyToken           = ("ReceiveOnly"            / "RC")
ReplyToken              = ("Reply"                  / "P")
ResetEventsDescriptorToken
                        = ("ResetEventsDescriptor"  / "RSE")
RestartToken            = ("Restart"                / "RS")
RemoteToken             = ("Remote"                 / "R")
RequestIDToken          = ("SPARequestID"           / "SPARQ")
ReservedGroupToken      = ("ReservedGroup"          / "RG")
ReservedValueToken      = ("ReservedValue"          / "RV")
SegmentationCompleteToken
                        = ("END"                    / "&")
SendonlyToken           = ("SendOnly"               / "SO")
SendrecvToken           = ("SendReceive"            / "SR")
ServicesToken           = ("Services"               / "SV")
ServiceStatesToken      = ("ServiceStates"          / "SI")
ServiceChangeIncompleteToken
                        = ("ServiceChangeInc"       / "SIC")
ServiceChangeToken      = ("ServiceChange"          / "SC")
ServiceChangeAddressToken
                        = ("ServiceChangeAddress"   / "AD")
SignalListToken         = ("SignalList"             / "SL")
```

```
SignalsToken             = ("Signals"                 / "SG")
SignalTypeToken          = ("SignalType"              / "SY")
StatsToken               = ("Statistics"              / "SA")
StreamToken              = ("Stream"                  / "ST")
SubtractToken            = ("Subtract"                / "S")
SynchISDNToken           = ("SynchISDN"               / "SN")
TerminationStateToken    = ("TerminationState"        / "TS")
TestToken                = ("Test"                    / "TE")
TimeOutToken             = ("TimeOut"                 / "TO")
TopologyToken            = ("Topology"                / "TP")
TransToken               = ("Transaction"             / "T")
ResponseAckToken         = ("TransactionResponseAck" / "K")
V18Token                 = ("V18")
V22Token                 = ("V22")
V22bisToken              = ("V22b")
V32Token                 = ("V32")
V32bisToken              = ("V32b")
V34Token                 = ("V34")
V76Token                 = ("V76")
V90Token                 = ("V90")
V91Token                 = ("V91")
VersionToken             = ("Version"                 / "V")
```

B.3 Hexadecimal octet coding

Hexadecimal octet coding is a means for representing a string of octets as a string of hexadecimal digits, with two digits representing each octet. This octet encoding should be used when encoding octet strings in the text version of the protocol.

For each octet, the 8-bit sequence is encoded as two hexadecimal digits. Bit 0 is the first transmitted; bit 7 is the last.

Bits 7-4 are encoded as the first hexadecimal digit, with Bit 7 as MSB and Bit 4 as LSB. Bits 3-0 are encoded as the second hexadecimal digit, with Bit 3 as MSB and Bit 0 as LSB.

Examples:

Octet bit pattern	Hexadecimal coding
00011011	D8
11100100	27
10000011 10100010 11001000 00001001	C1451390

B.4 Hexadecimal octet sequence

A hexadecimal octet sequence is an even number of hexadecimal digits, terminated by a <CR> character.

Annex C

Tags for media stream properties

Parameters for Local, Remote and LocalControl Descriptors are specified as tag-value pairs if binary encoding is used for the protocol. This annex contains the property names (PropertyID), the tags (Property tag), type of the property (Type) and the values (Value). Values presented in the Value field when the field contains references shall be regarded as "information". The reference contains the normative values. If a value field does not contain a reference, then the values in that field can be considered as "normative".

Referencing of Annex C properties follows the PackageID/PropertyID structure; however Annex C is not in itself a package. Annex C is considered to have PackageID 0x0000 for binary encoding and "anxc" for text encoding. For text encoding of H.248.1, Annex C shall only be used if the required property is either not defined by a package or not represented by SDP. The nesting of one Annex C property inside another is forbidden.

Tags are given as hexadecimal numbers in this annex. When setting the value of a property, a MGC may underspecify the value according to one of the mechanisms specified in 7.1.1.

It is optional to support the properties in this annex or any of its subclauses. For example, only three properties from C.3 and only five properties from C.8 might be implemented.

For type "enumeration" the value is represented by the value in brackets, e.g., Send (0), Receive (1). Annex C properties with the types "N bits" or "M Octets" should be treated as octet strings when encoding the protocol. Properties with "N bit integer" shall be treated as an integers. "String" shall be treated as an IA5String when encoding the protocol.

When a type is smaller than one octet, the value shall be stored in the low-order bits of an octet string of size one octet.

C.1 General media attributes

PropertyID	Property tag	Type	Value
Media	1001	Enumeration	Audio(0), Video(1), Data(2)
Transmission mode	1002	Enumeration	Send(0), Receive(1), Send&Receive(2)
Number of Channels	1003	Unsigned integer	0-255
Sampling rate	1004	Unsigned integer	0-2^32
Bitrate	1005	Integer	(0..4294967295) NOTE – Units of 100 bit/s.
ACodec	1006	Octet string	Audio Codec Type: Ref.: ITU-T Rec. Q.765.5 Non-ITU-T codecs are defined with the appropriate standards organization under a defined Organizational Identitifier.
Samplepp	1007	Unsigned integer	Maximum samples or frames per packet: 0..65535
Silencesupp	1008	Boolean	Silence Suppression: True/False
Encrypttype	1009	Octet string	Ref.: ITU-T Rec. H.245

PropertyID	Property tag	Type	Value
Encryptkey	100A	Octet string size (0..65535)	Encryption key Ref.: ITU-T Rec. H.235.0
Echocanc	100B		Not Used. See E.13 for an example of possible Echo Control Properties.
Gain	100C		Not Used. See E.13 for an available gain property.
Jitterbuff	100D	Unsigned integer	Jitter buffer size in ms: 0..65535
PropDelay	100E	Unsigned integer	Propagation Delay: 0..65535 Maximum propagation delay in milliseconds for the bearer connection between two media gateways. The maximum delay will be dependent on the bearer technology.
RTPpayload	100F	Integer	Payload type in RTP Profile for Audio and Video Conferences with Minimal Control Ref.: RFC 1890
Ptime	1010	Integer	Packetization Time This gives the length of time in milliseconds represented by the media in a packet. Ref.: IETF RFC 2327

C.2 Mux properties

PropertyID	Property tag	Type	Value
H222	2001	Octet string	H222LogicalChannelParameters Ref.: ITU-T Rec. H.245
H223	2002	Octet string	H223LogicalChannelParameters Ref.: ITU-T Rec. H.245
V76	2003	Octet string	V76LogicalChannelParameters Ref.: ITU-T Rec. H.245
H2250	2004	Octet string	H2250LogicalChannelParameters Ref.: ITU-T Rec. H.245

C.3 General bearer properties

PropertyID	Property tag	Type	Value
Mediatx	3001	Enumeration	Media Transport Type TDM Circuit(0), ATM(1), FR(2), Ipv4(3), Ipv6(4), …
BIR	3002	4 octets	Value depends on transport technology
NSAP	3003	1-20 octets	See NSAP. Ref.: Annex A/X.213

C.4 General ATM properties

PropertyID	Property tag	Type	Value
AESA	4001	20 octets	ATM End System Address
VPVC	4002	4 octets: VPCI in first two least significant octets, VCI in second two octets	VPCI/VCI Ref.: ITU-T Rec. Q.2931
SC	4003	Enumeration	Service Category: CBR(0), nrt-VBR1(1), nrt-VBR2(2), nrt-VBR3(3), rt-VBR1(4), rt-VBR2(5), rt-VBR3(6), UBR1(7), UBR2(8), ABR(9). Ref.: ATM Forum UNI 4.0
BCOB	4004	5-bit integer	Broadband Bearer Class Ref.: ITU-T Rec. Q.2961.2
BBTC	4005	7-bit integer	Broadband Transfer Capability Ref.: ITU-T Rec. Q.2961.1
ATC	4006	Enumeration	I.371 ATM Traffic Capability DBR(0), SBR1(1), SBR2(2), SBR3(3), ABT/IT(4), ABT/DT(5), ABR(6) Ref.: ITU-T Rec. I.371
STC	4007	2 bits	Susceptibility to clipping: Bits 2 1 0 0 not susceptible to clipping 0 1 susceptible to clipping Ref.: ITU-T Rec. Q.2931
UPCC	4008	2 bits	User Plane Connection configuration: Bits 2 1 0 0 point-to-point 0 1 point-to-multipoint Ref.: ITU-T Rec. Q.2931
PCR0	4009	24-bit integer	Peak Cell Rate (For CLP = 0) Ref.: ITU-T Rec. Q.2931
SCR0	400A	24-bit integer	Sustainable Cell Rate (For CLP = 0) Ref.: ITU-T Rec. Q.2961.1
MBS0	400B	24-bit integer	Maximum Burst Size (For CLP = 0) Ref.: ITU-T Rec. Q.2961.1
PCR1	400C	24-bit integer	Peak Cell Rate (For CLP = 0 + 1) Ref.: ITU-T Rec. Q.2931
SCR1	400D	24-bit integer	Sustainable Cell Rate (For CLP = 0 + 1) Ref.: ITU-T Rec. Q.2961.1

PropertyID	Property tag	Type	Value	
MBS1	400E	24-bit integer	Maximum Burst Size (For CLP = 0 + 1) Ref.: ITU-T Rec. Q.2961.1	
BEI	400F	Boolean	Best Effort Indicator Value 1 indicates that BEI is to be included in the ATM signalling; value 0 indicates that BEI is not to be included in the ATM signalling. Ref.: ATM Forum UNI 4.0	
TI	4010	Boolean	Tagging Indicator Value 0 indicates that tagging is not allowed; value 1 indicates that tagging is requested. Ref.: ITU-T Rec. Q.2961.1	
FD	4011	Boolean	Frame Discard Value 0 indicates that no frame discard is allowed; value 1 indicates that frame discard is allowed. Ref.: ATM Forum UNI 4.0	
A2PCDV	4012	24-bit integer	Acceptable 2-point CDV Ref.: ITU-T Rec. Q.2965.2	
C2PCDV	4013	24-bit integer	Cumulative 2-point CDV Ref.: ITU-T Rec. Q.2965.2	
APPCDV	4014	24-bit integer	Acceptable P-P CDV Ref.: ATM Forum UNI 4.0	
CPPCDV	4015	24-bit integer	Cumulative P-P CDV Ref.: ATM Forum UNI 4.0	
ACLR	4016	8-bit integer	Acceptable Cell Loss Ratio Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0	
MEETD	4017	16-bit integer	Maximum End-to-end transit delay Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0	
CEETD	4018	16-bit integer	Cumulative End-to-end transit delay Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0	
QosClass	4019	Integer 0-5	QoS Class	
			QoS Class	Meaning
			0	Default QoS associated with the ATC as defined in ITU-T Rec. Q.2961.2
			1	Stringent
			2	Tolerant
			3	Bi-level
			4	Unbounded
			5	Stringent Bi-level
			Ref.: ITU-T Rec. Q.2965.1	

PropertyID	Property tag	Type	Value
AALtype	401A	1 octet	AAL Type Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 AAL for voice 0 0 0 0 0 0 0 1 AAL type 1 0 0 0 0 0 0 1 0 AAL type 2 0 0 0 0 0 0 1 1 AAL type 3/4 0 0 0 0 0 1 0 1 AAL type 5 0 0 0 1 0 0 0 0 user-defined AAL Ref.: ITU-T Rec. Q.2931

C.5 Frame relay

PropertyID	Property tag	Type	Value
DLCI	5001	Unsigned integer	Data link connection ID
CID	5002	Unsigned integer	sub-channel ID
SID/Noiselevel	5003	Unsigned integer	silence insertion descriptor
Primary Payload type	5004	Unsigned integer	Primary Payload Type Covers FAX and codecs

C.6 IP

PropertyID	Property tag	Type	Value
IPv4	6001	32 bits Ipv4Address	IPv4Address Ref.: IETF RFC 791
IPv6	6002	128 bits	IPv6 Address Ref.: IETF RFC 2460
Port	6003	Unsigned integer	0..65535
Porttype	6004	Enumerated	TCP(0), UDP(1), SCTP(2)
RtcpbwRS	6005	Integer	RS RTCP bandwidth modifier indicates the RTCP bandwidth allocated to active data senders (as defined by the RTP specification) Ref.: IETF RFC 3556
RtcpbwRR	6006	Integer	RR RTCP bandwidth modifier indicates the RTCP bandwidth allocated to other participants in the RTP session (i.e., receivers) Ref.: IETF RFC 3556

C.7 ATM AAL 2

PropertyID	Property tag	Type	Value
AESA	7001	20 octets	AAL 2 service endpoint address as defined in the referenced Recommendation. ESEA NSEA Ref.: ITU-T Rec. Q.2630.1
BIR	See C.3	4 octets	Served user-generated reference as defined in the referenced Recommendation. SUGR Ref.: ITU-T Rec. Q.2630.1
ALC	7002	12 octets	AAL 2 link characteristics as defined in the referenced Recommendation. Maximum/Average CPS-SDU bit rate; Maximum/Average CPS-SDU size Ref.: ITU-T Rec. Q.2630.1
SSCS	7003	I.366.2: Audio (8 octets); Multirate (3 octets), or I.366.1: SAR-assured (14 octets); SAR-unassured (7 octets).	Service specific convergence sublayer information as defined in: – ITU-T Rec. Q.2630.1, and used in: – ITU-T Rec. I.366.2: Audio/Multirate; – ITU-T Rec. I.366.1: SAR-assured/-unassured. Ref.: ITU-T Recs Q.2630.1, I.366.1 and I.366.2
SUT	7004	1..254 octets	Served user transport parameter as defined in the referenced Recommendation. Ref.: ITU-T Rec. Q.2630.1
TCI	7005	Boolean	Test connection indicator as defined in the referenced Recommendation. Ref.: ITU-T Rec. Q.2630.1
Timer_CU	7006	32-bit integer	Timer-CU Milliseconds to hold partially filled cell before sending.
MaxCPSSDU	7007	8-bit integer	Maximum Common Part Sublayer Service Data Unit Ref.: ITU-T Rec. Q.2630.1
CID	7008	8 bits	Subchannel ID: 0-255 Ref.: ITU-T Rec. I.363.2

C.8 ATM AAL 1

PropertyID	Property tag	Type	Value
BIR	See table in C.3	4-29 octets	GIT (Generic Identifier Transport) Ref.: ITU-T Rec. Q.2941.1
AAL1ST	8001	1 octet	AAL 1 Subtype Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 null 0 0 0 0 0 0 0 1 voiceband signal transport on 64 kbit/s 0 0 0 0 0 0 1 0 circuit transport 0 0 0 0 0 1 0 0 high-quality audio signal transport 0 0 0 0 0 1 0 1 video signal transport Ref.: ITU-T Rec. Q.2931
CBRR	8002	1 octet	CBR Rate Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 1 64 kbit/s 0 0 0 0 0 1 0 0 1544 kbit/s 0 0 0 0 0 1 0 1 6312 kbit/s 0 0 0 0 0 1 1 0 32 064 kbit/s 0 0 0 0 0 1 1 1 44 736 kbit/s 0 0 0 0 1 0 0 0 97 728 kbit/s 0 0 0 1 0 0 0 0 2048 kbit/s 0 0 0 1 0 0 0 1 8448 kbit/s 0 0 0 1 0 0 1 0 34 368 kbit/s 0 0 0 1 0 0 1 1 139 264 kbit/s 0 1 0 0 0 0 0 0 n × 64 kbit/s 0 1 0 0 0 0 0 1 n × 8 kbit/s Ref.: ITU-T Rec. Q.2931
MULT	See table in C.9		Multiplier, or n × 64k/8k/300 Ref.: ITU-T Rec. Q.2931
SCRI	8003	1 octet	Source Clock Frequency Recovery Method Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 null 0 0 0 0 0 0 0 1 SRTS 0 0 0 0 0 0 1 0 ACM Ref.: ITU-T Rec. Q.2931
ECM	8004	1 octet	Error Correction Method Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 null 0 0 0 0 0 0 0 1 FEC – Loss 0 0 0 0 0 0 1 0 FEC – Delay Ref.: ITU-T Rec. Q.2931
SDTB	8005	16-bit integer	Structured Data Transfer Blocksize Block size of SDT CBR service Ref.: ITU-T Rec. I.363.1

PropertyID	Property tag	Type	Value
PFCI	8006	8-bit integer	Partially filled cells identifier 1-47 Ref.: ITU-T Rec. I.363.1

C.9 Bearer capabilities

The table entries referencing ITU-T Rec. Q.931 refer to the encoding in the bearer capability information element of Q.931, not to the low layer information element.

PropertyID	Property tag	Type	Value
TMR	9001	1 octet	Transmission Medium Requirement (ITU-T Rec. Q.763) Bits 8 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 speech 0 0 0 0 0 0 0 1 spare 0 0 0 0 0 0 1 0 64 kbit/s unrestricted 0 0 0 0 0 0 1 1 3.1 kHz audio 0 0 0 0 0 1 0 0 reserved for alternate speech (service 2)/64 kbit/s unrestricted (service 1) 0 0 0 0 0 1 0 1 reserved for alternate 64 kbit/s unrestricted (service 1)/speech (service 2) 0 0 0 0 0 1 1 0 64 kbit/s preferred 0 0 0 0 0 1 1 1 2 × 64 kbit/s unrestricted 0 0 0 0 1 0 0 0 384 kbit/s unrestricted 0 0 0 0 1 0 0 1 1536 kbit/s unrestricted 0 0 0 0 1 0 1 0 1920 kbit/s unrestricted 0 0 0 0 1 0 1 1 through 0 0 0 0 1 1 1 1 spare 0 0 0 1 0 0 0 0 3 × 64 kbit/s unrestricted 0 0 0 1 0 0 0 1 4 × 64 kbit/s unrestricted 0 0 0 1 0 0 1 0 5 × 64 kbit/s unrestricted 0 0 0 1 0 0 1 1 spare 0 0 0 1 0 1 0 0 7 × 64 kbit/s unrestricted 0 0 0 1 0 1 0 1 8 × 64 kbit/s unrestricted 0 0 0 1 0 1 1 0 9 × 64 kbit/s unrestricted 0 0 0 1 0 1 1 1 10 × 64 kbit/s unrestricted 0 0 0 1 1 0 0 0 11 × 64 kbit/s unrestricted 0 0 0 1 1 0 0 1 12 × 64 kbit/s unrestricted 0 0 0 1 1 0 1 0 13 × 64 kbit/s unrestricted 0 0 0 1 1 0 1 1 14 × 64 kbit/s unrestricted 0 0 0 1 1 1 0 0 15 × 64 kbit/s unrestricted 0 0 0 1 1 1 0 1 16 × 64 kbit/s unrestricted 0 0 0 1 1 1 1 0 17 × 64 kbit/s unrestricted 0 0 0 1 1 1 1 1 18 × 64 kbit/s unrestricted 0 0 1 0 0 0 0 0 19 × 64 kbit/s unrestricted 0 0 1 0 0 0 0 1 20 × 64 kbit/s unrestricted 0 0 1 0 0 0 1 0 21 × 64 kbit/s unrestricted 0 0 1 0 0 0 1 1 22 × 64 kbit/s unrestricted 0 0 1 0 0 1 0 0 23× 64 kbit/s unrestricted 0 0 1 0 0 1 0 1 spare

PropertyID	Property tag	Type	Value
			0 0 1 0 0 1 1 0 25 × 64 kbit/s unrestricted 0 0 1 0 0 1 1 1 26 × 64 kbit/s unrestricted 0 0 1 0 1 0 0 0 27 × 64 kbit/s unrestricted 0 0 1 0 1 0 0 1 28 × 64 kbit/s unrestricted 0 0 1 0 1 0 1 0 29 × 64 kbit/s unrestricted 0 0 1 0 1 0 1 1 through spare 1 1 1 1 1 1 1 1 Ref.: ITU-T Rec. Q.763
TMRSR	9002	1 octet	Transmission Medium Requirement Subrate 0 unspecified 1 8 kbit/s 2 16 kbit/s 3 32 kbit/s
Contcheck	9003	Boolean	Continuity Check 0 continuity check not required on this circuit 1 continuity check required on this circuit Ref.: ITU-T Rec. Q.763
ITC	9004	5 bits	Information Transfer Capability Bits 5 4 3 2 1 0 0 0 0 0 Speech 0 1 0 0 0 Unrestricted digital information 0 1 0 0 1 Restricted digital information 1 0 0 0 0 3.1 kHz audio 1 0 0 0 1 Unrestricted digital information with tones/announcements 1 1 0 0 0 Video All other values are reserved. Ref.: ITU-T Rec. Q.763
TransMode	9005	2 bits	Transfer Mode Bits 2 1 0 0 Circuit mode 1 0 Packet mode Ref.: ITU-T Rec. Q.931
TransRate	9006	5 bits	Transfer Rate Bits 5 4 3 2 1 0 0 0 0 0 This code shall be used for packet mode calls 1 0 0 0 0 64 kbit/s 1 0 0 0 1 2 × 64 kbit/s 1 0 0 1 1 384 kbit/s 1 0 1 0 1 1536 kbit/s 1 0 1 1 1 1920 kbit/s 1 1 0 0 0 Multirate (64 kbit/s base rate) Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
MULT	9007	7 bits	Rate Multiplier Any value from 2 to n (maximum number of B-channels) Ref.: ITU-T Rec. Q.931
layer1prot	9008	5 bits	User Information Layer 1 Protocol Bits 5 4 3 2 1 0 0 0 0 1 ITU-T standardized rate adaption V.110 and X.30. 0 0 0 1 0 ITU-T Rec. G.711 μ-law 0 0 0 1 1 ITU-T Rec. G.711 A-law 0 0 1 0 0 ITU-T Rec. G.726 32 kbit/s ADPCM and ITU-T Rec. I.460 0 0 1 0 1 ITU-T Recs H.221 and H.242 0 0 1 1 0 ITU-T Recs H.223 and H.245 0 0 1 1 1 Non-ITU-T standardized rate adaption. 0 1 0 0 0 ITU-T standardized rate adaption V.120. 0 1 0 0 1 ITU-T standardized rate adaption X.31 HDLC flag stuffing All other values are reserved. Ref.: ITU-T Rec. Q.931
syncasync	9009	Boolean	Synchronous/Asynchronous 0 Synchronous data 1 Asynchronous data Ref.: ITU-T Rec. Q.931
negotiation	900A	Boolean	Negotiation 0 In-band negotiation possible 1 In-band negotiation not possible Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
Userrate	900B	5 bits	User Rate Bits 5 4 3 2 1 0 0 0 0 0 Rate is indicated by E-bits specified in ITU-T Rec. I.460 or may be negotiated in-band 0 0 0 0 1 0.6 kbit/s ITU-T Recs V.6 and X.1 0 0 0 1 0 1.2 kbit/s ITU-T Rec. V.6 0 0 0 1 1 2.4 kbit/s ITU-T Recs V.6 and X.1 0 0 1 0 0 3.6 kbit/s ITU-T Rec. V.6 0 0 1 0 1 4.8 kbit/s ITU-T Recs V.6 and X.1 0 0 1 1 0 7.2 kbit/s ITU-T Rec. V.6 0 0 1 1 1 8 kbit/s ITU-T Rec. I.460 0 1 0 0 0 9.6 kbit/s ITU-T Recs V.6 and X.1 0 1 0 0 1 14.4 kbit/s ITU-T Rec. V.6 0 1 0 1 0 16 kbit/s ITU-T Rec. I.460 0 1 0 1 1 19.2 kbit/s ITU-T Rec. V.6 0 1 1 0 0 32 kbit/s ITU-T Rec. I.460 0 1 1 0 1 38.4 kbit/s ITU-T Rec. V.110 0 1 1 1 0 48 kbit/s ITU-T Recs V.6 and X.1 0 1 1 1 1 56 kbit/s ITU-T Rec. V.6 1 0 0 1 0 57.6 kbit/s ITU-T Rec. V.14 extended 1 0 0 1 1 28.8 kbit/s ITU-T Rec. V.110 1 0 1 0 0 24 kbit/s ITU-T Rec. V.110 1 0 1 0 1 0.1345 kbit/s ITU-T Rec. X.1 1 0 1 1 0 0.100 kbit/s ITU-T Rec. X.1 1 0 1 1 1 0.075/1.2 kbit/s ITU-T Recs V.6 and X.1 1 1 0 0 0 1.2/0.075 kbit/s ITU-T Recs V.6 and X.1 1 1 0 0 1 0.050 kbit/s ITU-T Recs V.6 and X.1 1 1 0 1 0 0.075 kbit/s ITU-T Recs V.6 and X.1 1 1 0 1 1 0.110 kbit/s ITU-T Recs V.6 and X.1 1 1 1 0 0 0.150 kbit/s ITU-T Recs V.6 and X.1 1 1 1 0 1 0.200 kbit/s ITU-T Recs V.6 and X.1 1 1 1 1 0 0.300 kbit/s ITU-T Recs V.6 and X.1 1 1 1 1 1 12 kbit/s ITU-T Rec. V.6 All other values are reserved. Ref.: ITU-T Rec. Q.931
INTRATE	900C	2 bits	Intermediate Rate Bits 2 1 0 0 Not used 0 1 8 kbit/s 1 0 16 kbit/s 1 1 32 kbit/s Ref.: ITU-T Rec. Q.931
nictx	900D	Boolean	Network Independent Clock (NIC) on transmission 0 Not required to send data with network independent clock 1 Required to send data with network independent clock Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
nicrx	900E	Boolean	Network independent clock (NIC) on reception 0 Cannot accept data with network independent clock (i.e., sender does not support this optional procedure) 1 Can accept data with network independent clock (i.e., sender does support this optional procedure) Ref.: ITU-T Rec. Q.931
flowconttx	900F	Boolean	Flow Control on transmission (Tx) 0 Not required to send data with flow control mechanism 1 Required to send data with flow control mechanism Ref.: ITU-T Rec. Q.931
flowcontrx	9010	Boolean	Flow control on reception (Rx) 0 Cannot accept data with flow control mechanism (i.e., sender does not support this optional procedure) 1 Can accept data with flow control mechanism (i.e., sender does support this optional procedure) Ref.: ITU-T Rec. Q.931
rateadapthdr	9011	Boolean	Rate adaption header/no header 0 Rate adaption header not included 1 Rate adaption header included Ref.: ITU-T Rec. Q.931
multiframe	9012	Boolean	Multiple frame establishment support in data link 0 Multiple frame establishment not supported. Only UI frames allowed 1 Multiple frame establishment supported Ref.: ITU-T Rec. Q.931
OPMODE	9013	Boolean	Mode of operation 0 Bit transparent mode of operation 1 Protocol sensitive mode of operation Ref.: ITU-T Rec. Q.931
llidnegot	9014	Boolean	Logical link identifier negotiation 0 Default, LLI = 256 only 1 Full protocol negotiation Ref.: ITU-T Rec. Q.931
assign	9015	Boolean	Assignor/assignee 0 Message originator is "default assignee" 1 Message originator is "assignor only" Ref.: ITU-T Rec. Q.931
inbandneg	9016	Boolean	In-band/out-band negotiation 0 Negotiation is done with USER INFORMATION messages on a temporary signalling connection 1 Negotiation is done in-band using logical link zero Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
stopbits	9017	2 bits	Number of stop bits Bits 2 1 0 0 Not used 0 1 1 bit 1 0 1.5 bits 1 1 2 bits Ref.: ITU-T Rec. Q.931
databits	9018	2 bits	Number of data bits excluding parity bit if present Bits 2 1 0 0 Not used 0 1 5 bits 1 0 7 bits 1 1 8 bits Ref.: ITU-T Rec. Q.931
parity	9019	3 bits	Parity information Bits 3 2 1 0 0 0 Odd 0 1 0 Even 0 1 1 None 1 0 0 Forced to 0 1 0 1 Forced to 1 All other values are reserved. Ref.: ITU-T Rec. Q.931
duplexmode	901A	Boolean	Mode duplex 0 Half duplex 1 Full duplex Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
modem	901B	6 bits	Modem Type Bits 6 5 4 3 2 1 0 0 0 0 0 0 through National use 0 0 0 1 0 1 0 1 0 0 0 1 ITU-T Rec. V.21 0 1 0 0 1 0 ITU-T Rec. V.22 0 1 0 0 1 1 ITU-T Rec. V.22 *bis* 0 1 0 1 0 0 ITU-T Rec. V.23 0 1 0 1 0 1 ITU-T Rec. V.26 0 1 1 0 0 1 ITU-T Rec. V.26 *bis* 0 1 0 1 1 1 ITU-T Rec. V.26 *ter* 0 1 1 0 0 0 ITU-T Rec..V.27 0 1 1 0 0 1 ITU-T Rec. V.27 *bis* 0 1 1 0 1 0 ITU-T Rec. V.27 *ter* 0 1 1 0 1 1 ITU-T Rec. V.29 0 1 1 1 0 1 ITU-T Rec. V.32 0 1 1 1 1 0 ITU-T Rec. V.34 1 0 0 0 0 0 through National use 1 0 1 1 1 1 1 1 0 0 0 0 through User specified 1 1 1 1 1 1 Ref.: ITU-T Q.931
layer2prot	901C	5 bits	User information layer 2 protocol Bits 5 4 3 2 1 0 0 0 1 0 ITU-T Rec. Q.921/I.441 0 0 1 1 0 ITU-T Rec. X.25, link layer 0 1 1 0 0 LAN logical link control (ISO/IEC 8802-2) All other values are reserved. Ref.: ITU-T Rec. Q.931
layer3prot	901D	5 bits	User information layer 3 protocol Bits 5 4 3 2 1 0 0 0 1 0 ITU-T Rec. Q.931 0 0 1 1 0 ITU-T Rec. X.25, packet layer 0 1 0 1 1 ISO/IEC TR 9577 (Protocol identification in the network layer) All other values are reserved. Ref.: ITU-T Rec. Q.931

PropertyID	Property tag	Type	Value
addlayer3prot	901E	Octet	Additional User Information layer 3 protocol Bits Bits 4 3 2 1 4 3 2 1 1 1 0 0 1 1 0 0 Internet Protocol (RFC 791) (ISO/IEC TR 9577) 1 1 0 0 1 1 1 1 Point-to-point Protocol (RFC 1661) Ref.: ITU-T Rec. Q.931
DialledN	901F	30 octets	Dialled Number
DiallingN	9020	30 octets	Dialling Number
ECHOCI	9021		Not Used. See E.13 for an example of possible Echo Control Properties.
NCI	9022	1 octet	Nature of Connection Indicators Bits 2 1 *Satellite Indicator* 0 0 no satellite circuit in the connection 0 1 one satellite circuit in the connection 1 0 two satellite circuits in the connection 1 1 spare Bits 4 3 *Continuity check indicator* 0 0 continuity check not required 0 1 continuity check required on this circuit 1 0 continuity check performed on a previous circuit 1 1 spare Bit 5 *Echo control device indicator* 0 outgoing echo control device not included 1 outgoing echo control device included Bits 8 7 6 Spare Ref.: ITU-T Rec. Q.763
USI	9023	Octet string	User Service Information Ref.: 3.57/Q.763

C.10 AAL 5 properties

PropertyID	Property tag	Type	Value
FMSDU	A001	32-bit integer	Forward Maximum CPCS-SDU Size: Maximum CPCS-SDU size sent in the direction from the calling user to the called user. Ref.: ITU-T Rec. Q.2931
BMSDU	A002	32-bit integer	Backwards Maximum CPCS-SDU Size: Maximum CPCS-SDU size sent in the direction from the called user to the calling user. Ref.: ITU-T Rec. Q.2931
SSCS	See table in C.7	See table in C.7	See table in C.7 Additional values: VPI/VCI

C.11 SDP equivalents

The SDP equivalents are subject to the SDP exceptions outlined in 7.1.8 for the text encoding of the protocol. For example, the CHOOSE wildcard is allowed to be used in the MGC to MG direction irrespective of the encoding (binary or text) of the protocol.

PropertyID	Property tag	Type	Value
SDP_V	B001	String	Protocol Version Ref.: RFC 2327
SDP_O	B002	String	Owner/creator and session ID Ref.: RFC 2327
SDP_S	B003	String	Session name Ref.: RFC 2327
SDP_I	B004	String	Session identifier Ref.: RFC 2327
SDP_U	B005	String	URI of descriptor Ref.: RFC 2327
SDC_E	B006	String	email address Ref.: RFC 2327
SDP_P	B007	String	phone number Ref.: RFC 2327
SDP_C	B008	String	Connection information Ref.: RFC 2327
SDP_B	B009	String	Bandwidth Information Ref.: RFC 2327
SDP_Z	B00A	String	Time zone adjustment Ref.: RFC 2327
SDP_K	B00B	String	Encryption Key Ref.: RFC 2327

PropertyID	Property tag	Type	Value
SDP_A	B00C	String	Zero or more session attributes Ref.: RFC 2327
SDP_T	B00D	String	Active Session Time Ref.: RFC 2327
SDP_R	B00E	String	Zero or more repeat times Reference: RFC 2327
SDP_M	B00F	String	Media type, port, transport and format Ref.: RFC 2327

C.12 H.245

PropertyID	Property tag	Type	Value
OLC	C001	Octet string	The value of H.245 OpenLogicalChannel structure. Ref.: ITU-T Rec. H.245
OLCack	C002	Octet string	The value of H.245 OpenLogicalChannelAck structure. Ref.: ITU-T Rec. H.245
OLCcnf	C003	Octet string	The value of H.245 OpenLogicalChannelConfirm structure. Ref.: ITU-T Rec. H.245
OLCrej	C004	Octet string	The value of H.245 OpenLogicalChannelReject structure. Ref.: ITU-T Rec. H.245
CLC	C005	Octet string	The value of H.245 CloseLogicalChannel structure. Ref.: ITU-T Rec. H.245
CLCack	C006	Octet string	The value of H.245 CloseLogicalChannelAck structure. Ref.: ITU-T Rec. H.245
LCN	C007	Integer	The value of H.245 Local Channel Number 0-65535. Ref.: ITU-T Rec. H.245

Annex D

Transport over IP

D.1 Transport over IP/UDP using Application Level Framing (ALF)

Protocol messages defined in this Recommendation may be transmitted over UDP. When no port is provided by the peer (see 7.2.8), commands should be sent to the default port number: 2944 for text-encoded operation, or 2945 for binary-encoded operation. Responses must be sent to the address and port from which the corresponding commands were sent.

ALF is a set of techniques that allows an application, as opposed to a stack, to affect how messages are sent to the other side. A typical ALF technique is to allow an application to change the order of messages sent when there is a queue after it has queued them. There is no formal specification for ALF. The procedures in Annex D.1 contain a minimum suggested set of ALF behaviours.

Implementors using IP/UDP with ALF should be aware of the restrictions of the MTU on the maximum message size.

D.1.1 Providing at-most-once functionality

Messages carried over UDP may be subject to losses. In the absence of a timely response, commands are repeated. Most commands are not idempotent. The state of the MG would become unpredictable if, for example, Add Commands were executed several times. The transmission procedures shall thus provide an "At-Most-Once" functionality.

Peer protocol entities are expected to keep in memory a list of the responses that they sent to recent transactions and a list of the transactions that are currently outstanding. The TransactionID of each incoming message is compared to the TransactionIDs of the recent responses sent to the same MID. If a match is found, the entity does not execute the transaction, but simply repeats the response. If no match is found, the message will be compared to the list of currently outstanding transactions. If a match is found in that list, indicating a duplicate transaction, the entity does not execute the transaction (see D.1.4 for procedures on sending TransactionPending).

The procedure uses a long timer value, noted LONG-TIMER in the following. The timer should be set larger than the maximum duration of a transaction, which should take into account the maximum number of repetitions, the maximum value of the repetition timer and the maximum propagation delay of a packet in the network. A suggested value is 30 seconds.

The copy of the responses may be destroyed either LONG-TIMER seconds after the response is issued, or when the entity receives a confirmation that the response has been received, through the "Response Acknowledgement parameter". For transactions that are acknowledged through this parameter, the entity shall keep a copy of the TransactionID for LONG-TIMER seconds after the response is issued, in order to detect and ignore duplicate copies of the transaction request that could be produced by the network.

D.1.2 Transaction identifiers and three-way handshake

D.1.2.1 Transaction identifiers

TransactionIDs are 32-bit integer numbers. A Media Gateway Controller may decide to use a specific number space for each of the MGs that they manage, or to use the same number space for all MGs that belong to some arbitrary group. MGCs may decide to share the load of managing a large MG between several independent processes. These processes will share the same transaction number space. There are multiple possible implementations of this sharing, such as having a centralized allocation of TransactionIDs, or pre-allocating non-overlapping ranges of identifiers to different processes. The implementations shall guarantee that unique TransactionIDs are allocated to all transactions that originate from a logical MGC (identical MID). MGs can simply detect duplicate transactions by looking at the TransactionID and MID only.

D.1.2.2 Three-way handshake

The TransactionResponseAcknowledgement parameter can be found in any message. It carries a set of "confirmed TransactionID ranges". Entities may choose to delete the copies of the responses to transactions whose ID is included in "confirmed TransactionID ranges" received in the transaction response messages. They should silently discard further commands when the TransactionID falls within these ranges.

The "confirmed TransactionID ranges" values shall not be used if more than LONG-TIMER seconds have elapsed since the MG issued its last response to that MGC, or when a MG resumes operation. In this situation, transactions should be accepted and processed, without any test on the TransactionID.

Messages that carry the "TransactionResponseAcknowledgement" parameter may be transmitted in any order. The entity shall retain the "confirmed TransactionID ranges" received for LONG-TIMER seconds.

In the binary encoding, if only the `firstAck` is present in a response acknowledgement (see A.2), only one transaction is acknowledged. If both `firstAck` and `lastAck` are present, then the range of transactions from `firstAck` to `lastAck` is acknowledged. In the text encoding, a horizontal dash is used to indicate a range of transactions being acknowledged (see B.2).

D.1.3 Computing retransmission timers

It is the responsibility of the requesting entity to provide suitable timeouts for all outstanding transactions, and to retry transactions when timeouts have been exceeded. Furthermore, when repeated transactions fail to be acknowledged, it is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending connections.

The specification purposely avoids specifying any value for the retransmission timers. These values are typically network-dependent. The retransmission timers should normally estimate the timer value by measuring the time spent between the sending of a command and the return of a response. Implementations shall ensure that the algorithm used to calculate retransmission timing performs an exponentially increasing backoff of the retransmission timeout for each retransmission or repetition after the first one.

NOTE – One possibility is to use the algorithm implemented in TCP-IP, which uses two variables:

- The average acknowledgement delay (AAD), estimated through an exponentially smoothed average of the observed delays.
- The average deviation (ADEV), estimated through an exponentially smoothed average of the absolute value of the difference between the observed delay and the current average. The retransmission timer, in TCP, is set to the sum of the average delay plus N times the average deviation. The maximum value of the timer should however be bounded for the protocol defined in this Recommendation, in order to guarantee that no repeated packet would be received by the gateways after LONG-TIMER seconds. A suggested maximum value is 4 seconds.

After any retransmission, the entity should do the following:

- It should double the estimated value of the average delay, AAD.
- It should compute a random value, uniformly distributed between 0.5 AAD and AAD.
- It should set the retransmission timer to the sum of that random value and N times the average deviation.

This procedure has two effects. Because it includes an exponentially increasing component, it will automatically slow down the stream of messages in case of congestion. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

D.1.4 Provisional responses

Executing some transactions may require a long time. Long execution times may interact with the timer-based retransmission procedure. This may result either in an inordinate number of retransmissions, or in timer values that become too long to be efficient. Entities that can predict that a transaction will require a long execution time may send a provisional response, "TransactionPending". They should send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a TransactionPending shall switch to a different repetition timer for repeating requests. Upon receipt of a final response following receipt of provisional responses, an immediate confirmation shall be sent, and normal repetition timers shall be used thereafter. An entity that sends a provisional response shall include the immAckRequired field in the ensuing final response,

indicating that an immediate confirmation is expected. Receipt of a TransactionPending after receipt of a reply shall be ignored.

D.1.5 Repeating requests, responses and acknowledgements

The protocol is organized as a set of transactions, each of which is composed of a request and a response, commonly referred to as an acknowledgement. The protocol messages, being carried over UDP, may be subject to losses. In the absence of a timely response, transactions are repeated. Entities are expected to keep in memory a list of the responses that they sent to recent transactions, i.e., a list of all the responses they sent over the last LONG-TIMER seconds, and a list of the transactions that are currently being executed.

The repetition mechanism is used to guard against three types of possible errors:

- transmission errors, when, for example, a packet is lost due to noise on a line, or congestion in a queue;
- component failure, when, for example, an interface to a entity becomes unavailable;
- entity failure, when, for example, an entire entity becomes unavailable.

The entities should be able to derive from the past history an estimate of the packet loss rate due to transmission errors. In a properly configured system, this loss rate should be kept very low, typically less than 1%. If a Media Gateway Controller or a Media Gateway has to repeat a message more than a few times, it is legitimate to assume that something else than a transmission error is occurring. For example, given a loss rate of 1%, the probability that five consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a Media Gateway Controller that processes 1000 transactions per second. (Indeed, the number of repetitions that is considered excessive should be a function of the prevailing packet loss rate.) We should note that the "suspicion threshold", which we will call "Max1", is normally lower than the "disconnection threshold", which should be set to a larger value.

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after retransmitting the packet an excessive number of times (typically between 7 and 11 times). In order to account for the possibility of an undetected or in-progress "failover", we modify the classic algorithm so that if the Media Gateway receives a valid ServiceChange message announcing a failover, it will start transmitting outstanding commands to that new MGC. Responses to commands are still transmitted to the source address of the command.

In order to automatically adapt to network load, this Recommendation specifies exponentially increasing timers. If the initial timer is set to 200 milliseconds, the loss of a fifth retransmission will be detected after about 6 seconds. This is probably an acceptable waiting delay to detect a failover. The repetitions should continue after that delay not only in order to perhaps overcome a transient connectivity problem, but also in order to allow some more time for the execution of a failover (waiting a total delay of 30 seconds is probably acceptable).

It is, however, important that the maximum delay of retransmissions be bounded. Prior to any retransmission, it should be checked that the time elapsed since the sending of the initial datagram is no greater than T-MAX. If more than T-MAX time has elapsed, the MG concludes that the MGC has failed, and that it begins its recovery process as described in 11.5. If the MG retries to connect to the current MGC, it shall use a ServiceChange with ServiceChangeMethod set to Disconnected so that the new MGC will be aware that the MG lost one or more transactions. The value T-MAX is related to the LONG-TIMER value: the LONG-TIMER value is obtained by adding to T-MAX the maximum propagation delay in the network.

D.2 Using TCP

Protocol messages as defined in this Recommendation may be transmitted over TCP. When no port is specified by the other side (see 7.2.8) the commands should be sent to the default port. The defined protocol has messages as the unit of transfer, while TCP is a stream-oriented protocol. TPKT, according to RFC 1006, shall be used to delineate messages within the TCP stream.

In a transaction-oriented protocol, there are still ways for transaction requests or responses to be lost. As such, it is recommended that entities using TCP transport implement application level timers for each request and each response, similar to those specified for application level framing over UDP.

D.2.1 Providing the at-most-once functionality

Messages carried over TCP are not subject to transport losses, but loss of a transaction request or its reply may nonetheless be noted in real implementations. In the absence of a timely response, commands are repeated. Most commands are not idempotent. The state of the MG would become unpredictable if, for example, Add Commands were executed several times.

To guard against such losses, it is recommended that entities follow the procedures in D.1.1.

D.2.2 Transaction identifiers and three-way handshake

For the same reasons, it is possible that transaction replies may be lost even with a reliable delivery protocol such as TCP. It is recommended that entities follow the procedures in D.1.2.2.

D.2.3 Computing retransmission timers

With reliable delivery, the incidence of loss of a transaction request or reply is expected to be very low. Therefore, only simple timer mechanisms are required. Exponential back-off algorithms should not be necessary, although they could be employed where, as in an MGC, the code to do so is already required since MGCs must implement ALF/UDP as well as TCP.

D.2.4 Provisional responses

As with UDP, executing some transactions may require a long time. Entities that can predict that a transaction will require a long execution time may send a provisional response, "TransactionPending". They should send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a TransactionPending shall switch to a longer repetition timer for that transaction.

Entities shall retain transactions and replies until they are confirmed. The basic procedure of D.1.4 should be followed, but simple timer values should be sufficient. There is no need to send an immediate confirmation upon receipt of a final response.

D.2.5 Ordering of commands

TCP provides ordered delivery of transactions. No special procedures are required. It should be noted that ALF/UDP allows the sending entity to modify its behaviour under congestion, and in particular, could reorder transactions when congestion is encountered. TCP could not achieve the same results.

Annex E

Basic packages

This annex contains definitions of some packages for use with this Recommendation.

E.1 Generic

Package Name: Generic

PackageID: g (0x0001)

Description: Generic package for commonly encountered items

Version: 2

Extends: None

E.1.1 Properties

None

E.1.2 Events

E.1.2.1 Cause

Event Name: Cause

EventID: cause (0x0001)

Description: Generic error event

EventsDescriptor Parameters: None

ObservedEvents Descriptor Parameters:

General Cause

Parameter Name: General Cause

ParameterID: Generalcause (0x0001)

Description: This parameter groups the failures into six groups, which the MGC may act upon.

Type: enumeration

Optional: No

Possible values:

"NR"	(0x0001)	Normal	Release
"UR"	(0x0002)	Unavailable	Resources
"FT"	(0x0003)	Failure,	Temporary
"FP"	(0x0004)	Failure,	Permanent
"IW"	(0x0005)	Interworking	Error
"UN"	(0x0006)	Unsupported	

Default: None

Failure Cause

Parameter Name: Failure Cause

ParameterID: Failurecause (0x0002)

Description: The Failure Cause is the value generated by the Released equipment, i.e., a released network connection. The concerned value is defined in the appropriate bearer control protocol.

Type: OCTET STRING

Optional: Yes

Possible values: OCTET STRING

Default: None

E.1.2.2 Signal Completion

Event Name: Signal Completion

EventID: sc (0x0002)

Description: Indicates the termination of a signal for which the NotifyCompletion parameter was set to enable reporting of a completion event. For further procedural description, see 7.1.1, 7.1.11 and 7.2.7.

EventsDescriptor Parameters: None

ObservedEvents Descriptor Parameters:

Signal Identity

Parameter Name: Signal Identity

ParameterID: SigID (0x0001)

Description: This parameter identifies the signal which has terminated. For a signal that is contained in a SignalList, the SignalListID parameter should also be returned indicating the appropriate list.

Type: Binary: octet (string), Text: string

Optional: No

Possible values: a signal which has terminated. A signal shall be identified using the pkgdName syntax without wildcarding.

Default: None

Termination Method

Parameter Name: Termination Method

ParameterID: Meth (0x0002)

Description: Indicates the means by which the signal terminated.

Type: Enumeration

Optional: No

Possible values:

"TO" (0x0001) Signal timed out or otherwise completed on its own
"EV" (0x0002) Interrupted by event
"SD" (0x0003) Halted by new Signals Descriptor
"NC" (0x0004) Not completed, other cause
"PI" (0x0005) First to penultimate iteration. For last iteration, use TO.

Default: None

Signal List ID

Parameter Name: Signal List ID

ParameterID: SLID (0x0003)

Description: Indicates to which SignalList a signal belongs. The SignalListID is only returned in cases where the signal resides in a SignalList.

Type: Integer

Optional: Yes (only supported if signal lists are used)

Possible values: 1-65535

Default: None

Request ID

Parameter Name: Request ID

ParameterID: RID (0x0004)

Description: Indicates to which NotifyCompletion request the SignalID is associated.

Type: Integer

Optional: Yes (only supported if signal lists are used)

Possible values: 1-4294967295

Default: None

E.1.3 Signals

None

E.1.4 Statistics

None

E.2 Base Root Package

Package Name: Base Root Package

PackageID: root (0x0002)

Description: This package defines Gateway wide properties.

Version: 2

Extends: None

E.2.1 Properties

E.2.1.1 Maximum Number of Contexts

Property Name: MaxNrOfContexts

PropertyID: maxNumberOfContexts (0x0001)

Description: The value of this property gives the maximum number of contexts that can exist at any time. The NULL Context is not included in this number.

Type: Double

Possible values: 1 and up

Default: Provisioned

Defined in: TerminationState

Characteristics: ReadOnly

E.2.1.2 Maximum Terminations Per Context

Property Name: MaxTerminationsPerContext

PropertyID: maxTerminationsPerContext (0x0002)

Description: The maximum number of allowed terminations in a context, see 6.1.

Type: Integer

Possible values: any positive integer

Default: Provisioned

Defined in: TerminationState

Characteristics: ReadOnly

E.2.1.3 Normal MG Execution Time

Property Name: normalMGExecutionTime

PropertyID: normalMGExecutionTime (0x0003)

Description: Settable by the MGC to indicate the interval within which the MGC expects a response to any transaction from the MG (exclusive of network delay).

Type: Integer

Possible values: any positive integer; represents milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.1.4 Normal MGC Execution Time

Property Name: normalMGCExecutionTime

PropertyID: normalMGCExecutionTime (0x0004)

Description: Settable by the MGC to indicate the interval within which the MG should expect a response to any transaction from the MGC (exclusive of network delay).

Type: Integer

Possible values: any positive integer; represents milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.1.5 MG Provisional Response Timer Value

Property Name: MGProvisionalResponseTimerValue

PropertyID: MGProvisionalResponseTimerValue (0x0005)

Description: Indicates the time within which the MGC should expect a Pending Response from the MG if a transaction cannot be completed. Initially set to normalMGExecutionTime plus network delay, but may be lowered.

Type: Integer

Possible Values: any positive integer; represents milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.1.6 MGC Provisional Response Timer Value

Property Name: MGCProvisionalResponseTimerValue

PropertyID: MGCProvisionalResponseTimerValue (0x0006)

Description: Indicates the time within which the MG should expect a Pending Response from the MGC if a transaction cannot be completed. Initially set to normalMGCExecutionTime plus network delay, but may be lowered.

Type: Integer

Possible Values: any positive integer; represents milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.1.7 MGC Originated Pending Limit

Property Name: MGCOriginatedPendingLimit

PropertyID: MGCOriginatedPendingLimit (0x0007)

Description: Indicates the number of TransactionPendings that can be received from the MGC. Once this limit is exceeded, the MGC should issue a TransactionReply with Error 506 ("Number of TransactionPendings Exceeded"); otherwise, the MG can assume the transaction to be in error.

Type: Integer

Possible Values: any positive integer

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.1.8 MG Originated Pending Limit

Property Name: MGOriginatedPendingLimit

PropertyID: MGOriginatedPendingLimit (0x0008)

Description: Indicates the number of TransactionPendings that can be received from the MG. Once this limit is exceeded, the MG should issue a TransactionReply with Error 506 ("Number of TransactionPendings Exceeded"); otherwise, the MGC can assume the transaction to be in error.

Type: Integer

Possible Values: any positive integer

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.2.2 Events

None.

E.2.3 Signals

None.

E.2.4 Statistics

None.

E.2.5 Procedures

None.

E.3 Tone Generator Package

Package Name: Tone Generator Package

PackageID: tonegen (0x0003)

Description: This package defines signals to generate audio tones. This package does not specify parameter values. It is intended to be extendable. Generally, tones are defined as an individual signal with a parameter, ind, representing "interdigit" time delay, and a tone ID to be used with playtone. A tone ID should be kept consistent with any tone generation for the same tone. MGs are expected to be provisioned with the characteristics of appropriate tones for the country in which the MG is located.

Version: 2

Designed to be extended only: Yes

Extends: None

E.3.1 Properties

None

E.3.2 Events

None

E.3.3 Signals

E.3.3.1 Play Tone

Signal Name: Play Tone

SignalID: pt (0x0001)

Description: Plays audio tone over an audio channel

Signal Type: Brief

Duration: Provisioned

Additional parameters:

Tone ID list

Parameter Name: Tone ID List

ParameterID: tl (0x0001)

Description: List of tones to be played in sequence.

Type: Sublist of Enumeration

Optional: No

Possible Values: The sublist shall contain one or more tone IDs.

Default: None

Inter-signal duration

Parameter Name: Inter-signal duration

ParameterID: ind (0x0002)

Description: Timeout between two consecutive tones in milliseconds

Type: Integer

Optional: Yes

Possible Values: Any integer greater than or equal to 0

Default: Provisioned

Tone Direction

Parameter Name: Tone Direction

ParameterID: btd (0x0003)

Description: Direction the tone should progress from the termination.

Type: Enumeration

Optional: Yes

Possible values: "EXT" (0x0001) External
"INT" (0x0002) Internal
"BOTH" (0x0003) Both

Default: External

E.3.4 Statistics

None

E.3.5 Procedures

No tone IDs are specified in this package. Packages that extend this package can add possible values for tone ID as well as adding individual tone signals.

E.4 Tone Detection Package

Package Name: Tone Detection Package

PackageID: tonedet (0x0004)

Description: This package defines events for audio tone detection. Tones are selected by name (tone ID). MGs are expected to be provisioned with the characteristics of appropriate tones for the country in which the MG is located.

This package does not specify parameter values. It is intended to be extendable.

Version: 1

Designed to be extended only: Yes

Extends: None

E.4.1 Properties

None

E.4.2 Events

E.4.2.1 Start Tone Detected

Event Name: Start tone detected

EventID: std (0x0001)

Description: Detects the start of a tone. The characteristics of positive tone detection are implementation dependent.

EventsDescriptor Parameters:

Tone ID List

Parameter Name: Tone ID List

ParameterID: tl (0x0001)

Type: Sublist of Enumeration

Optional: No

Possible values: The only tone ID defined in this package is "wild card" which is "*" in text encoding and 0x0000 in binary. Extensions to this package would add possible values for tone ID. If tl is "wildcard", any tone ID is detected.

Default: None

ObservedEventsDescriptor Parameters:

Tone ID

Parameter Name: Tone ID

ParameterID: tid (0x0003)

Description: The tone ID of the detected tone.

Type: Enumeration

Optional: No

Possible values: "wildcard" as defined above is the only value defined in this package. Extensions to this package would add additional possible values for tone ID.

Default: none

E.4.2.2 End Tone Detected

Event Name: End tone detected

EventID: etd (0x0002)

Description: Detects the end of a tone.

EventsDescriptor Parameters:

Tone ID List

Parameter Name: Tone ID List

ParameterID: tl (0x0001)

Description: A list of tone IDs to be detected.

Type: Sublist of enumeration

Optional: No

Possible values: No possible values are specified in this package. Extensions to this package would add possible values for tone ID.

Default: None

ObservedEventsDescriptor Parameters:

Tone ID

Parameter Name: Tone ID

ParameterID: tid (0x0003)

Description: The tone ID of the detected tone.

Type: Enumeration

Optional: No

Possible values: "wildcard" as defined above is the only value defined in this package. Extensions to this package would add possible values for tone ID.

Default: None

Duration

Parameter Name: Duration

ParameterID: dur (0x0002)

Description: This parameter contains the duration of the tone from first detection until it stopped.

Type: Integer

Optional: Yes

Possible values: Any positive integer in milliseconds

Default: None

E.4.2.3 Long Tone Detected

Event Name: Long Tone Detected

EventID: ltd (0x0003)

Description: Detects that a tone has been playing for at least a certain amount of time

EventsDescriptor Parameters:

Tone ID List

Parameter Name: Tone ID List

ParameterID: tl (0x0001)

Description: A list of tone IDs to be detected.

Type: Sublist of Enumeration

Optional: No

Possible values: "wildcard" as defined above is the only value defined in this package. Extensions to this package would add possible values for tone ID.

Default: None

Duration

Parameter Name: Duration

ParameterID: dur (0x0002)

Description: The duration to test against.

Type: Integer

Optional: Yes

Possible values: any positive integer, expressed in milliseconds

Default: Provisioned

ObservedEventsDescriptor Parameters:

Tone ID

Parameter Name: Tone ID

ParameterID: tid (0x0003)

Description: The tone ID of the detected tone.

Type: Enumeration

Optional: No

Possible values: No possible values are specified in this package. Extensions to this package would add possible values for tone ID.

Default: None

E.4.3 Signals

None

E.4.4 Statistics

None

E.4.5 Procedures

None

E.5 Basic DTMF Generator Package

Package Name: Basic DTMF Generator Package

PackageID: dg (0x0005)

Description: This package defines the basic DTMF tones as signals and extends the allowed values of parameter tl of playtone in tonegen.

Version: 2

Extends: tonegen version 2

E.5.1 Properties

None

E.5.2 Events

None

E.5.3 Signals

E.5.3.1 DTMF Character 0

Signal Name: DTMF Character 0

SignalID: d0 (0x0010)

Description: Generate DTMF 0 tone. The physical characteristic of DTMF 0 is defined in the gateway.

Signal Type: Brief

Duration: Provisioned

Additional parameters:

Tone Direction

Parameter Name: Tone Direction

ParameterID: btd (0x0001)

Description: Direction the tone should progress from the termination.

Type: Enumeration

Optional: Yes

Possible values: "EXT" (0x0001) External
"INT" (0x0002) Internal
"BOTH" (0x0003) Both

Default: External

Additional values: d0 (0x0010) is defined as a tone ID for playtone.

The other DTMF characters are specified in exactly the same way. A table with all signal names and SignalIDs is included. Note that each DTMF character is defined as both a signal and a tone ID, thus extending the basic tone generation package. Also note that DTMF SignalIDs are different from the names used in a DigitMap.

Signal name	SignalID/Tone ID
DTMF character 0	d0 (0x0010)
DTMF character 1	d1 (0x0011)
DTMF character 2	d2 (0x0012)
DTMF character 3	d3 (0x0013)
DTMF character 4	d4 (0x0014)
DTMF character 5	d5 (0x0015)
DTMF character 6	d6 (0x0016)
DTMF character 7	d7 (0x0017)
DTMF character 8	d8 (0x0018)
DTMF character 9	d9 (0x0019)
DTMF character *	ds (0x0020)

Signal name	SignalID/Tone ID
DTMF character #	do (0x0021)
DTMF character A	da (0x001a)
DTMF character B	db (0x001b)
DTMF character C	dc (0x001c)
DTMF character D	dd (0x001d)

E.5.4 Statistics

None

E.5.5 Procedures

None

E.6 DTMF Detection Package

Package Name: DTMF Detection Package

PackageID: dd (0x0006)

Description: This package defines the basic DTMF tones detection. This package extends the possible values of tone ID in the Start Tone Detected, End Tone Detected and Long Tone Detected Events.

Additional tone ID values are all tone IDs described in package dg (Basic DTMF Generator Package).

The following table maps DTMF events to DigitMap symbols as described in 7.1.14.

DTMF	Event Symbol
d0	"0"
d1	"1"
d2	"2"
d3	"3"
d4	"4"
d5	"5"
d6	"6"
d7	"7"
d8	"8"
d9	"9"
da	"A" or "a"
db	"B" or "b"
dc	"C" or "c"
dd	"D" or "d"
ds	"E" or "e"
do	"F" or "f"

Version: 1

Extends: tonedet version 1

E.6.1 Properties

None

E.6.2 Events

E.6.2.1 DTMF Digits

Event Name: DTMF Digits

EventID: EventIDs are defined with the same names as the SignalIDs defined in the table found in E.5.3.

Description: Generated when the MG detects a digit.

EventDescriptor Parameters: None

ObservedEvents Descriptor Parameters: None

E.6.2.2 DigitMap Completion Event

Event Name: DigitMap Completion Event

EventID: ce (0x0004)

Description: Generated when a DigitMap completes as described in 7.1.14.

EventsDescriptor Parameters: None

ObservedEventsDescriptor Parameters:

DigitString

Parameter Name: DigitString

ParameterID: ds (0x0001)

Description: the portion of the current dial string as described in 7.1.14 which matched part or all of an alternative event sequence specified in the DigitMap.

Type: String of DigitMap symbols (possibly empty) returned as a quotedString

Optional: No

Possible values: a sequence of the characters "0" through "9", "A" through "F", and the long duration modifier "Z".

Default: None

Termination Method

Parameter Name: Termination Method

ParameterID: Meth (0x0003)

Description: indicates the reason for generation of the event. See the procedures in 7.1.14.

Type: Enumeration

Optional: No

Possible values:

"UM" (0x0001)	Unambiguous match
"PM" (0x0002)	Partial match, completion by timer expiry or unmatched event
"FM" (0x0003)	Full match, completion by timer expiry or unmatched event

Default: None

E.6.3 Signals

None

E.6.4 Statistics

None

E.6.5 Procedures

DigitMap processing is activated only if an Events Descriptor is activated that contains a DigitMap completion event as defined in E.6.2 and that DigitMap completion event contains an EventDM field in the requested actions as defined in 7.1.9. Other parameters such as KeepActive or embedded Events or Signals Descriptors may also be present in the Events Descriptor and do not affect the activation of DigitMap processing.

E.7 Call Progress Tones Generator Package

Package Name: Call Progress Tones Generator Package

PackageID: cg 0x0007

Description: This package defines the basic call progress tones as signals and extends the allowed values of the tl parameter of playtone in tonegen.

Version: 2

Extends: tonegen version 2

E.7.1 Properties

None

E.7.2 Events

None

E.7.3 Signals

E.7.3.1 Dial Tone

Signal Name: Dial Tone

SignalID: dt (0x0030)

Description: Generate dial tone. The physical characteristic of dial tone is available in the gateway.

Signal Type: TimeOut

Duration: Provisioned

Additional parameters: None

Additional values: dt (0x0030) is defined as a tone ID for playtone

The other tones of this package are defined in exactly the same way. A table with all signal names and SignalIDs is included. Note that each tone is defined as both a signal and a tone ID, thus extending the basic tone generation package.

Signal Name	SignalID/tone ID
Dial Tone	dt (0x0030)
Ringing Tone	rt (0x0031)
Busy Tone	bt (0x0032)
Congestion Tone	ct (0x0033)
Special Information Tone	sit (0x0034)
(Recording) Warning Tone	wt (0x0035)
Payphone Recognition Tone	prt (0x0036)
Call Waiting Tone	cw (0x0037)
Caller Waiting Tone	cr (0x0038)

E.7.4 Statistics

None

E.7.5 Procedures

NOTE – The required set of tone IDs corresponds to those defined in ITU-T Rec. E.180/Q.35. See ITU-T Rec. E.180/Q.35 for definition of the meanings of these tones.

E.8 Call progress tones detection package

Package Name: Call Progress Tones Detection Package

PackageID: cd (0x0008)

Description: This package defines the basic call progress detection tones. This package extends the possible values of tone ID in the Start Tone Detected, End Tone Detected and Long Tone Detected Events.

Additional values

Tone ID values are defined for Start Tone Detected, End Tone Detected and Long Tone Detected with the same values as those in package cg (Call Progress Tones Generation Package).

The required set of tone IDs corresponds to ITU-T Rec. E.180/Q.35. See ITU-T Rec. E.180/Q.35 for definition of the meanings of these tones.

Version: 1

Extends: tonedet version 1

E.8.1 Properties

None

E.8.2 Events

Events are defined as in the Call Progress Tones Generator Package (cg) for the tones listed in the table of E.7.3.

E.8.3 Signals

None

E.8.4 Statistics

None

E.8.5 Procedures

None

E.9 Analog Line Supervision Package

Package Name: Analog Line Supervision Package

PackageID: al, 0x0009

Description: This package defines events and signals for an analog line.

Version: 1

Extends: None

E.9.1 Properties

None

E.9.2 Events

E.9.2.1 On-hook

Event Name: On-hook

EventID: on (0x0004)

Description: Detects handset going on-hook. Whenever an Events Descriptor is activated that requests monitoring for an on-hook event and the line is already on-hook, then the MG shall behave according to the setting of the "strict" parameter.

EventsDescriptor Parameters:

Strict Transition

Parameter Name: Strict Transition

ParameterID: strict (0x0001)

Description: Indicates how the on-hook event is detected.

Type: Enumeration

Optional: Yes

Possible values: "exact" (0x0000) Only an actual hook state transition to on-hook is to be recognized

"state" (0x0001) The event is to be recognized either if the hook state transition is detected or if the hook state is already on-hook

"failWrong" (0x0002) If the hook state is already on-hook, the command fails and an error is reported

Default: Exact

ObservedEventsDescriptor Parameters:

Initial State

Parameter Name: Initial State

ParameterID: init (0x0002)

Description: The reason for reporting the on-hook transition. Only returned if the Strict Transition parameter is set to "state".

Type: Boolean

Possible values: "True" The line was already on-hook when the Events Descriptor containing this event was activated
"False" An actual state transition to on-hook was detected.

Default: None

E.9.2.2 Off-hook

Event Name: Off-hook

EventID: of (0x0005)

Description: Detects handset going off-hook. Whenever an Events Descriptor is activated that requests monitoring for an off-hook event and the line is already off-hook, then the MG shall behave according to the setting of the "strict" parameter.

EventsDescriptor Parameters:

Strict Transition

Parameter Name: Strict Transition

ParameterID: strict (0x0001)

Description: Indicates how the off-hook event is detected.

Type: Enumeration

Possible values: "exact" (0x0000) Only an actual hook state transition to off-hook is to be recognized

"state" (0x0001) The event is to be recognized either if the hook state transition is detected or if the hook state is already off-hook

"failWrong" (0x0002) If the hook state is already off-hook, the command fails and an error is reported

Default: Exact

ObservedEventsDescriptor Parameters:

Initial State

Parameter Name: Initial State

ParameterID: init (0x0002)

Description: The reason for reporting the off-hook transition. Only returned if the Strict Transition parameter is set to "state".

Type: Boolean

Optional: Yes

Possible values: "True" The line was already off-hook when the Events Descriptor containing this event was activated
"False" An actual state transition to off-hook was detected.

Default: None

E.9.2.3 Flashhook

Event Name: Flashhook

EventID: fl (0x0006)

Description: Detects handset flash. A flash occurs when an on-hook is followed by an off-hook between a minimum and maximum duration.

EventsDescriptor Parameters:

Minimum Duration

Parameter Name: Minimum Duration

ParameterID: mindur (0x0004)

Description: The minimum time between an on-hook and an off-hook to be detected as a flash.

Type: Integer

Optional: Yes

Possible values: Any positive integer in milliseconds

Default: Provisioned.

Maximum Duration

Parameter Name: Maximum Duration

ParameterID: maxdur (0x0005)

Description: The maximum time between an on-hook and an off-hook to be detected as a flash.

Type: Integer

Optional: Yes

Possible values: Any positive integer in milliseconds

Default: Provisioned.

ObservedEventsDescriptor Parameters: None

E.9.3 Signals

E.9.3.1 Ring

Signal Name: Ring

SignalID: ri (0x0002)

Description: Applies ringing on the line

Signal Type: TimeOut

Duration: Provisioned

Additional parameters:

Cadence

Parameter Name: Cadence

ParameterID: cad (0x0006)

Description: Represents durations of alternating on and off segments, constituting a complete ringing cycle starting with an on. Restricted function MGs may ignore cadence values they are incapable of generating.

Type: Sublist of Integer

Optional: Yes

Possible values: Any positive integer in milliseconds

Default: Provisioned

Frequency

Parameter Name: Frequency

ParameterID: freq (0x0007)

Description: The frequency of the ringing. Restricted function MGs may ignore frequency values they are incapable of generating.

Type: Integer

Optional: Yes

Possible values: Any positive integer in Hz

Default: Provisioned.

E.9.4 Statistics

None

E.9.5 Error codes

Error Code #: 540

Name: Unexpected initial hook state

Definition: This error is generated when the MGC has tried to request a hook state transition event with the Strict Parameter set to "failWrong", and the hook state is already what the transition implies.

Error Text in the Error Descriptor: –

Comment: –

E.9.6 Procedures

If the MGC sets an Events Descriptor containing a hook state transition event (on-hook or off-hook) with the Strict (0x0001) Parameter set to "failWrong", and the hook state is already what the transition implies, the execution of the command containing that Events Descriptor fails. The MG shall include Error Code 540 ("Unexpected initial hook state") in its response.

E.10 Basic Continuity Package

Package Name: Basic Continuity Package

PackageID: ct (0x000a)

Description: This package defines events and signals for continuity test. The continuity test includes provision of either a loopback or transceiver functionality.

Version: 1

Extends: None

E.10.1 Properties

None

E.10.2 Events

E.10.2.1 Continuity Test Completion

Event Name: Completion

EventID: cmp (0x0005)

Description: This event detects the completion of a continuity test.

EventsDescriptor Parameters: None

ObservedEventsDescriptor Parameters:

Continuity Test Result

Parameter Name: Result

ParameterID: res (0x0008)

Description: Indicates the result of the continuity test.

Type: Enumeration

Optional: No

Possible values: "SUCCESS" (0x0001) Success
"FAILURE" (0x0000) Failure

Default: None

E.10.3 Signals

E.10.3.1 Continuity Test

Signal Name: Continuity Test

SignalID: ct (0x0003)

Discussion: Initiates sending of continuity test tone on the termination to which it is applied.

Signal Type: TimeOut

Default: Provisioned

Additional parameters: None

E.10.3.2 Respond to Continuity Test

Signal Name: Respond

SignalID: rsp (0x0004)

Description: The signal is used to respond to a continuity test. See E.10.5 for further explanation.

Signal Type: On/Off

Default: Provisioned

Additional parameters: None

E.10.4 Statistics

None

E.10.5 Procedures

When a MGC wants to initiate a continuity test, it sends a command to the MG containing:

- a Signals Descriptor with the ct Signal; and
- an Events Descriptor containing the cmp Event.

Upon reception of a command containing the ct Signal and cmp Event, the MG initiates the continuity test tone for the specified termination. If the return tone is detected and any other required conditions are satisfied before the signal times out, the cmp Event shall be generated with the value of the result parameter equal to success. In all other cases, the cmp Event shall be generated with the value of the result parameter equal to failure.

When a MGC wants the MG to respond to a continuity test, it sends a command to the MG containing a Signals Descriptor with the rsp Signal. Upon reception of a command with the rsp Signal, the MG either applies a loopback or (for 2-wire circuits) awaits reception of a continuity test tone. In the loopback case, any incoming information shall be reflected back as outgoing information. In the 2-wire case, any time the appropriate test tone is received, the appropriate response tone should be sent. The MGC determines when to remove the rsp Signal.

When a continuity test is performed on a termination, no echo devices or codecs shall be active on that termination.

Performing voice path assurance as part of continuity testing is provisioned by bilateral agreement between network operators.

NOTE – Example tones and test procedure details are given in clauses 7 and 8/Q.724, 2.1.8/Q.764 and ITU-T Rec. Q.1902.4.

E.11 Network Package

Package Name: Network Package

PackageID: nt (0x000b)

Description: This package defines properties of network terminations independent of network type. This includes, but is not limited to, TDM, IP and ATM.

Version: 1

Extends: None

E.11.1 Properties

E.11.1.1 Maximum Jitter Buffer

Property Name: Maximum Jitter Buffer

PropertyID: jit (0x0007)

Description: This property puts a maximum size on the jitter buffer.

Type: Integer

Possible values: Any positive integer in milliseconds.

Default: Provisioned

Defined in: LocalControl Descriptor

Characteristics: Read/Write

E.11.2 Events

E.11.2.1 Network Failure

Event Name: Network Failure

EventID: netfail (0x0005)

Description: The termination generates this event upon detection of a failure due to external or internal network reasons.

EventsDescriptor Parameters: None

ObservedEventsDescriptor Parameters:

Cause

Parameter Name: Cause

ParameterID: cs (0x0001)

Description: This parameter may be included with the failure event to provide diagnostic information on the reason of failure.

Type: String

Optional: Yes

Possible values: any text string

Default: None

E.11.2.2 Quality Alert

Event Name: Quality Alert

EventID: qualert (0x0006)

Description: This event allows the MG to indicate a loss of quality of the network connection. The MG may do this by measuring packet loss, inter-arrival jitter, propagation delay and then indicating this using a percentage of quality loss.

EventsDescriptor Parameters:

Threshold

Parameter Name: Threshold

ParameterID: th (0x0001)

Description: threshold for percent of quality loss measured, calculated based on a provisioned method that could take into consideration packet loss, jitter, and delay for example. Event is triggered when calculation exceeds the threshold.

Type: Integer

Optional: Yes

Possible values: 0 to 99

Default: Provisioned

ObservedEventsDescriptor Parameters:

Threshold

Parameter Name: Threshold

ParameterID: th (0x0001)

Description: percent of quality loss measured, calculated based on a provisioned method that could take into consideration packet loss, jitter, and delay for example.

Type: Integer

Optional: Yes

Possible values: 0 to 99

Default: None

E.11.3 Signals

None

E.11.4 Statistics

E.11.4.1 Duration

Statistic Name: Duration

StatisticsID: dur (0x0001)

Description: provides duration of time the termination has existed or been out of the NULL context.

Type: Double

Possible values: Any positive integer in milliseconds

Level: Either

E.11.4.2 Octets Sent

Statistic Name: Octets Sent

StatisticID: os (0x0002)

Description: Provides the number of octets sent from the termination or stream since the termination has existed or been out of the NULL Context. The octets represent the egress media flow excluding all transport overhead. At the termination level, it is equal to the sum of the egress flows over all streams.

For analogue transmission media, the octet count shall be set equal to zero.

Type: Double

Possible values: any 64-bit integer 0 and up

Level: Either

E.11.4.3 Octets Received

Statistic Name: Octets Received

StatisticID: or (0x0003)

Type: Double

Description: Provides the number of octets received on the termination or stream since the termination has existed or been out of the NULL Context. The octets represent the ingress media flow excluding all transport overhead. At the termination level, it is equal to the sum of the ingress flows over all streams.

For analogue transmission media, the octet count shall be set equal to zero.

Possible values: any 64-bit integer 0 and up

Level: Either

E.11.5 Procedures

E.11.5.1 Procedures for application of the Network Failure and Quality Alert events

The application of the two events defined by the package depends on:

- the specific bearer type (e.g., TDM, Analog Line, RTP, IP, AAL 2, AAL 1, etc.);
- the underlying transmission layer technology (e.g., analog, PDH, SDH, SONET, xDSL, 802.3, 802.11, etc.);
- whether the event is associated with resources internal or external to the MG.

E.11.5.1.1 Common for both events

The application of either event requires an understanding between the MGC and the MG. If the MG and MGC lack a consistent view of their meaning, the MGC will not be able to act on them effectively.

Explicit recommendations are beyond the scope of the H.248.1 core protocol. The specific usage of these events should be rather specified in H.248 Profile specifications.

E.11.5.1.2 Specific to event "Network Failure"

The Cause Parameter can be used to distinguish between different failure modes. The termination type also helps in the interpretation of the event.

E.11.5.1.3 Specific to event "Quality Alert"

The detection logic should be based on one or more performance metrics related to quality of service from which a value comparable to the Threshold Parameter may be derived.

The details of the underlying measurement and estimation process (the QoS-related observation variables, the filter mechanisms, the update period lengths, etc.) may be specified in an H.248 Profile.

E.11.5.1.4 Relationship to ServiceChange procedures

A Network Failure or Quality Alert Event may trigger a ServiceChange procedure on the termination(s) affected. The transition to OutOfService may be initiated by the MG or the MGC, depending on circumstances.

E.12 RTP Package

Package Name: RTP Package

PackageID: rtp (0x000c)

Description: This package is used to support packet-based multimedia data transfer by means of the Real-time Transport Protocol (RTP) (RFC 3550).

Version: 1

Extends: nt version 1

E.12.1 Properties

None

E.12.2 Events

E.12.2.1 Payload Transition

Event Name: Payload Transition

EventID: pltrans (0x0001)

Description: This event detects and notifies when there is a transition of the RTP payload format from one format to another.

EventsDescriptor Parameters: None

ObservedEventsDescriptor Parameters:

RTP Payload Type

Parameter Name: rtppayload

ParameterID: rtppltype (0x01)

Description: The payload format to which the transition was made.

Type: Sublist of Enumeration

Optional: No

Possible values: The encoding method shall be specified by using one or several valid encoding names, as defined in the RTP AV Profile or registered with IANA.

Default: None

E.12.3 Signals

None

E.12.4 Statistics

E.12.4.1 Packets Sent

Statistic Name: Packets Sent

StatisticID: ps (0x0004)

Description: Provides the number of packets sent from the termination or stream since the termination has existed or been out of the NULL Context.

Type: Double

Possible values: any 64-bit integer 0 and up

Level: Either

E.12.4.2 Packets Received

Statistic Name: Packets Received

StatisticID: pr (0x0005)

Description: Provides the number of packets received on the termination or stream since the termination has existed or been out of the NULL Context.

Type: Double

Possible values: any 64-bit integer 0 and up

Level: Either

E.12.4.3 Packet Loss

Statistic Name: Packet Loss

StatisticID: pl (0x0006)

Description: Describes the current rate of packet loss on an RTP stream, as defined in RFC 3550. Packet loss is expressed as percentage value: number of packets lost in the interval between two reception reports, divided by the number of packets expected during that interval.

Type: Double

Possible values: a 32-bit whole number and a 32-bit fraction.

Level: Either

E.12.4.4 Jitter

Statistic Name: Jitter

StatisticID: jit (0x0007)

Description: Requests the current value of the inter-arrival jitter on an RTP stream as defined in RFC 3550. Jitter measures the variation in inter-arrival time for RTP data packets.

Type: Double

Possible values: any 64-bit integer 0 and up

Level: Either

E.12.4.5 Delay

Statistic Name: Delay

StatisticID: delay (0x0008)

Description: Requests the current value of packet propagation delay expressed in timestamp units. This is the same as average latency.

Type: Double

Possible values: any 64-bit integer 0 and up

Level: Either

E.12.5 Procedures

When RTCP is associated with an RTP stream, RTCP shall remain unaffected by the H.248.1 Mode Property in the LocalControl Descriptor.

When RTCP is associated with an RTP stream and the MG receives an Empty Remote Descriptor for that stream, the MG shall stop the RTCP stream along with the corresponding RTP stream.

E.13 TDM Circuit Package

Package Name: TDM Circuit Package

PackageID: tdmc (0x000d)

Description: This package may be used by any termination that supports gain and echo control. It was originally intended for use on TDM circuits but may be more widely used.

New versions or extensions of this package should take non-TDM use into account.

Version: 1

Extends: Network Package version 1

E.13.1 Properties

E.13.1.1 Echo Cancellation

Property Name: Echo Cancellation

PropertyID: ec (0x0008)

Type: Boolean

Possible values: "True" (when the echo cancellation is requested)
"False" (when it is turned off)

Default: Provisioned

Defined in: LocalControl Descriptor

Characteristics: Read/Write

E.13.1.2 Gain Control

Property Name: Gain Control

PropertyID: gain (0x000a)

Description: Gain control, or usage of signal level adaptation and noise level reduction, is used to adapt the level of the outbound signal. However, it is necessary, for example for modem calls, to turn off this function. When the value is set to "automatic", the termination serves as an automatic level control (ALC) with a target level provisioned on the MG and the direction being outward.

Type: Integer

Possible values: The gain control specifies the gain in decibels (positive or negative), with the maximum positive integer, 214748647 (0x7fffffff), reserved to represent "automatic".

Default: Provisioned

Defined in: LocalControl Descriptor

Characteristics: Read/Write

E.13.2 Events

None

E.13.3 Signals

None

E.13.4 Statistics

None

E.13.5 Procedures

None

E.14 Segmentation Package

Package Name: Segmentation Package

PackageID: seg (0x00A3)

Description: This package defines properties for use when performing H.248-based segmentation on non-segmenting transports.

Version: 1

Extends: root version 2

E.14.1 Properties

E.14.1.1 MG Segmentation Timer Value

Property Name: MGSegmentationTimerValue

PropertyID: MGSegmentationTimerValue (0x0009)

Description: Indicates the time within which the MGC should expect the reception of any outstanding message segments from the MG once the SegmentationCompleteToken is received. Initially set to MGProvisionalResponseTimerValue, but may be changed.

Type: Integer

Possible values: any positive integer in milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.14.1.2 MGC Segmentation Timer Value

Property Name: MGCSegmentationTimerValue

PropertyID: MGCSegmentationTimerValue (0x000A)

Description: Indicates the time within which the MG should expect the reception of any outstanding message segments from the MGC once the SegmentationCompleteToken is received. Initially set to MGCProvisionalResponseTimerValue, but may be changed.

Type: Integer

Possible values: any positive integer in milliseconds

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.14.1.3 MG Maximum PDU Size

Property Name: MGMaxPDUSize

PropertyID: MGMaxPDUSize (0x000B)

Description: Indicates the maximum size of the MG's incoming protocol data unit for the control association's transport protocol. The MGC should avoid building messages that exceed this size.

Type: Integer

Possible values: any positive integer in bytes

Default: Provisioned

Defined in: TerminationState

Characteristics: ReadOnly

E.14.1.4 MGC Maximum PDU Size

Property Name: MGCMaxPDUSize

PropertyID: MGCMaxPDUSize (0x000C)

Description: Indicates the maximum size of the MGC's incoming protocol data unit for the control association's transport protocol. The MG should avoid building messages that exceed this size.

Type: Integer

Possible values: any positive integer in bytes

Default: Provisioned

Defined in: TerminationState

Characteristics: Read/Write

E.14.2 Events

None

E.14.3 Signals

None

E.14.4 Statistics

None

E.14.5 Error Codes

Error Code #: **459**

Name: Segments not received

Definition: This error indicates that the recipient of a segmented TransactionReply timed out waiting for all the segments to be delivered.

Error Text in the Error Descriptor: The missing segment numbers are included.

Comment: –

E.14.6 Procedures

Because some transports do not automatically segment messages that exceed the maximum transmission unit of the transport, messages may end up being truncated. If the message in question contains multiple TransactionReplies, the sender should send each TransactionReply in a separate message. If there remain messages that exceed the MTU length, the sender may implement the following segmentation procedures for TransactionReplies. TransactionRequests shall not be segmented.

When segmenting a transaction reply, the sender shall ensure that the message contains complete commands and/or actions. Each segment is indicated by additional segment information at the transaction level. Each segment shall use the same transactionID, and shall not repeat commands or completed actions in subsequent segments. Each segment shall be numbered serially starting at 1 and continuing through the last segment, which is denoted by inclusion of the SegmentationCompleteToken. In all cases, the segmented messages shall be syntactically valid constructs. The receiver replies to each segment in turn, utilizing the segmentation response. Because the segments are complete messages in themselves, the receiving entity does not need to wait for further segments before processing any particular segment.

Upon reception of the final segment, the receiver shall determine if it has received all the message segments. If it has, then the receiver shall respond with a TransactionResponseAcknowledgement.

If the receiver has not received all the segments, it shall start a timer, and wait for any missing segments. The length of the timer shall be provisionable upon the receiver, but should be the same length as the ProvisionalResponseTimerValue (see E.2.1) of the other entity. If the missing segments do not arrive before the timer expires, the receiver shall reply with Error Code 459 ("Segments not received").

Example 1:

```
Sender:       !/3 [12.34.56.78]:2944 P=1/1{C=1{AV=term1{…}, AV=term2{…}}}
Receiver:     !/3 [12.34.56.79]:2944 SM=1/1
Sender:       !/3 [12.34.56.78]:2944 P=1/2{C=1{AV=term3{…}},C=2{AV=term4{…}}}
Receiver:     !/3 [12.34.56.79]:2944 SM=1/2
Sender:       !/3 [12.34.56.78]:2944 P=1/3/#{C=3{AV=term5{…}}}
Receiver:     !/3 [12.34.56.79]:2944 SM=1/3/#
Receiver:     !/3 [12.34.56.79]:2944 K=1
```

Example 2:

```
Sender:       !/3 [12.34.56.78]:2944 P=1/1{C=1{AV=term1{…}, AV=term2{…}}}
Receiver:     !/3 [12.34.56.79]:2944 SM=1/1
Sender:       !/3 [12.34.56.78]:2944 P=1/4/#{C=3{AV=term5{…}}}
Receiver:     !/3 [12.34.56.79]:2944 SM=1/4/#
```

```
/* Segmentation Timer Expires */
Receiver:     !/3 [12.34.56.79]:2944 ER=459{"2,3"}
```

E.15 Notification Behaviour

Package Name: Notification Behaviour Package

PackageID: nb (0x009a)

Description: The package has functionality that enables the MG at the request of the MGC to regulate the sending of Notify Commands. This package has an interaction with the NotifyBehaviour flag described in 7.1.9. This version of the package describes regulation behaviour based upon a percentage regulation. Other types of regulation behaviour are for further study.

Version: 1

Extends: None

E.15.1 Properties

E.15.1.1 Notification Regulation

Property Name: Notification Regulation

PropertyId: notreg (0x0001)

Description: Indicates the percentage of notifications on the MG that should be suppressed. A value of 0% indicates no suppression. A value of 100% indicates all notifies should be suppressed. This property shall be defined only on the Root Termination. When "Regulated Notify" is sent on a particular event, the MG uses the Notification Regulation Property to determine the correct percentage of Notifies to regulate (i.e., suppress).

Type: Integer

Possible values: 0-100

Default: None

Defined in: TerminationState

Characteristics: Read/Write

E.15.2 Events

None.

E.15.3 Signals

None

E.15.4 Statistics

None

E.15.5 Procedures

E.15.5.1 NotifyBehaviour

The NotifyBehaviour mechanism allows the MGC to manage how it should receive Notify.request commands from the MG.

E.15.5.1.1 NeverNotify

There may be instances where the MGC may not be interested in receiving a Notify.request command. For example: when detection of the occurrence of this event is essential to trigger an

embedded signal or event, but the MGC does not need to be informed. Setting NotifyBehaviour Parameter to "NeverNotify" on an event ensures that the detection of the event is not notified to the MGC; however, any embedded signals or events are executed. NeverNotify does not have any relation to the Notification Regulation Property (nb/notreg) and events so marked are not included in the regulation calculations.

E.15.5.1.2 NotifyImmediate

The MGC may want to be notified of important events such as an 'Emergency Call' event, even though some other events are being regulated. This is achieved by setting the NotifyBehaviour Parameter to "NotifyImmediate" for that particular event. This is the current default behaviour when events are detected. NotifyImmediate does not have any relation to the Notification Regulation Property (nb/notreg) and events so marked are not included in the regulation calculations.

E.15.5.1.3 RegulatedNotify

E.15.5.1.3.1 Percentage Based Regulated Notify

When the NotifyBehaviour Parameter is set to "NotifyRegulated" on a particular event, the decision to send to the MGC a notification for any particular event occurrence is governed by the Notification Regulation (nb/notreg) Property on the Root Termination. The value of this property is determined and set by the MGC. For example, the regulation rate could correspond to the MGC congestion level. The MG will then be responsible for determining the correct percentage of Notify.Commands to be suppressed amongst the events that are marked as "NotifyRegulated" across the entire MG.

The MG may implement any algorithm that achieves the set Notification Regulation percentage. The algorithm should be chosen so that the risk of synchronization between different MGs is minimized. A suggested algorithm is given below:

For a given regulation percentage (e.g., 10%) the MG would do the following for events that had the RegulatedNotify flag:

1) Suppress a number of notifications (e.g., Notification 1).
2) Accept a number of subsequent notifications until the NotificationRegulation percentage is reached (e.g., accept 9 Notifications).
3) Repeat steps 1 and 2 until the Regulation Percentage is modified.
4) To avoid problems with synchronization the MG should start in a random position in the suggested sequence above.

If an event has the NotifyBehaviour Parameter set to "RegulatedNotify" and the Notification Regulation Property is set to zero, this will have the same effect as if the event was set to "ImmediateNotify", which is the default behaviour for event notification.

The NotifyBehaviour mechanism should be used with caution to avoid incorrect states. For example, if the notification of an "off-hook" event is suppressed by the MG, then the MGC will not be able to provide the correct response (e.g., dial tone, congestion tone). This incorrect behaviour is avoided through the use of the embedded signal or embedded events mechanism to return an appropriate response without the MGC being involved at that time. The MGC may set both an embedded Signals or Events Descriptor to be triggered when an event marked "NotifyRegulated" is detected and accepted (i.e., notification not suppressed) and a separate regulated embedded Signals or Events Descriptor to be triggered when the event is detected and regulated (i.e., notification suppressed).

NOTE – Even in the above case the MG is still unaware of what constitutes a call. It is simply performing a pre-programmed task rather than call control functionality.

E.15.5.2 Example Scenario

Given the scenario where an MGC and an MG are terminating analogue lines, the MGC/MG may exhibit different behaviour based on whether the MGC is in an overload state or not and whether the analogue line (ALN) is originating an emergency call or not. The different behaviours are shown in Table E.1 below:

Table E.1/H.248.1 – Potential MGC/MG Overload Behaviour

MG Behaviour Task: "Handling of analogue line call signalling (ALS)" = call control signalling traffic for POTS terminals (ALN; analogue line interfaces)		**1) Call/Context Type**	
		Emergency call	Non-emergency call
2) MGC Load State NOTE – two-state model assumed.	Non-overload state (e.g.,) low, medium, high load)	I) MG: Normal ALS processing with *highest* priority	II) MG: Normal ALS processing with *lower* priority
	Overload state	III) MG: Normal ALS processing ==> Call control signalling forwarded to MGC in "usual H.248 mode"	IV) MG: Programmed local ALS handling ==> MGC *not* involved ==> Call protocol compliant rejected via "congestion tone"

E.15.5.2.1 Quadrant I and II Behaviour

No special NotifyBehaviour is required on the MG as the MGC is not in overload and can receive all messages (notifications) from the MG. However, once the MGC has determined the type of call, it may set a priority in the MG by using an appropriate Context Attribute.

E.15.5.2.2 Quadrant III and IV Behaviour

When the MGC has determined it is in an overloaded state, it may send the following command to all or some selected group of the terminations in the NULL Context.

```
Example message:
Transaction=1234{
   Context = - {
      Modify = aln/* {        ; select terminations as desired
         Events = 1234 {
            al/of {
               RegulatedNotify {
                  Embed {
                     Signals {cg/dt} ,
; If the off-hook is regulated apply dial tone.
                     Events = 1235 {
                         xdd/xce {
                            DigitMap = PriorityDialPlan1,
                            bc = 20,
                            mp = enhanced,
                            ImmediateNotify
                         },
; If emergency generate notify to MGC
                         xdd/xce {
                            DigitMap = NonPriorityDialPlan1,
                            bc = 20,
                            mp = enhanced,
```

```
                              NeverNotify,
                              Embed{
                                 Signals {cg/ct}
                              }
                           },
; If non-priority send congestion tone and never notify
                           al/on {
                              NeverNotify,
                              ResetEventsDescriptor
                           }
; If on-hook is detected then reset the EventsDescriptor
; and never notify
                        }
                     }
                  }
               }
            }
         }
      }
   }
}
```

NOTE – The context identity may be specific or NULL and the terminationID may be wildcarded or specific depending on how and when this behaviour is set on the MG. Use of Wildcard reply will limit the number of responses if the overload behaviour is loaded as the MGC goes into overload.

E.15.5.2.3 If the Notification Regulation Property is set to non-zero (overloaded state)

The behaviour of the MG in the case that an off-hook is detected and it is regulated would be to:

- suppress the notification of an off-hook to the MGC;
- apply dial tone;
- receive digits dialled by the subscriber;
- if it has been determined that the user has dialled an emergency number, always generate the notification of the match with the "priority dial plan";
- if it has been determined that the user has dialled a non-priority number, always suppress the notification of the match with the "non-priority dial plan" and apply congestion tone;
- enable the detection of on-hook (with NeverNotify) which on the receipt of an on-hook would cause the Events Descriptor on the termination to be reset back to the state before embedding was activated (or the MG-provisioned Events Descriptor).

E.15.5.2.4 If the Notification Regulation Property is set equal to zero (non-overloaded state)

The behaviour of the MG in the above case would be to:

- send a notification of off-hook to the MGC.

If the overload behaviour Events Descriptor is set using explicit Modify Commands from the MGC (as opposed to using MG provisioning), then whilst the MGC remains in the overloaded condition, the overload behaviour Events Descriptor will be reloaded for each termination as part of the clearing sequence of any call/call attempt notified to the MGC.

When the MGC overload condition has ceased, the MGC may reset terminations in the NULL Context back to default non-overload behaviour. Alternatively, the MGC should set the Notification Regulation property equal to zero.

Annex F

ServiceChange Procedures

F.1 Introduction

This annex describes the use of ServiceChange procedures when certain events occur on a MG or MGC. It serves to clarify the ServiceChange procedures as described in 7.2.8 and 11. Where there are discrepancies between this annex and this Recommendation, the procedures of the main body of this Recommendation, take precedence over those described in this annex.

The events leading to the sending of a ServiceChange Command described in this annex are:

1) MG Registration – The MG registers with a MGC and a control association is established.
2) MG Re-Registration – The MG registers with a MGC after it has been requested to do so by a MGC.
3) MG Service Cancellation – The MG informs the MGC that terminations or the MG as a whole is to be taken Out-of-Service.
4) MG Service Restoration – The MG has recovered after failure and notifies the MGC.
5) MG Redundant Takeover – A secondary MG takes over in the instance of failure or maintenance of a primary MG.
6) MG Lost Communication – The MG informs the MGC that it had previously lost communication with the MGC but this communication has now been restored.
7) MG Capability Change – The MG informs the MGC that the capabilities of terminations or the MG as a whole has changed.
8) MGC Initiated MG Re-Registration – The MGC informs the MG that the MG should re-register with the same MGC or another MGC.
9) MGC Initiated Service Restoration – The MGC has recovered after failure and orders the MG to place the relevant terminations in an initial state.
10) MGC Initiated Service Cancellation – The MGC informs the MG that terminations or the MG as a whole is being taken OutOfService.
11) MGC Redundant Failover – The primary MGC is failing and directs the MG to register with a specific secondary MGC.

Figure F.1 illustrates these ServiceChange triggering events, the entity in which the event appears and which event pairs are correlated. Figure F.2 shows a state model to which the MG and terminations on the MG will conform.

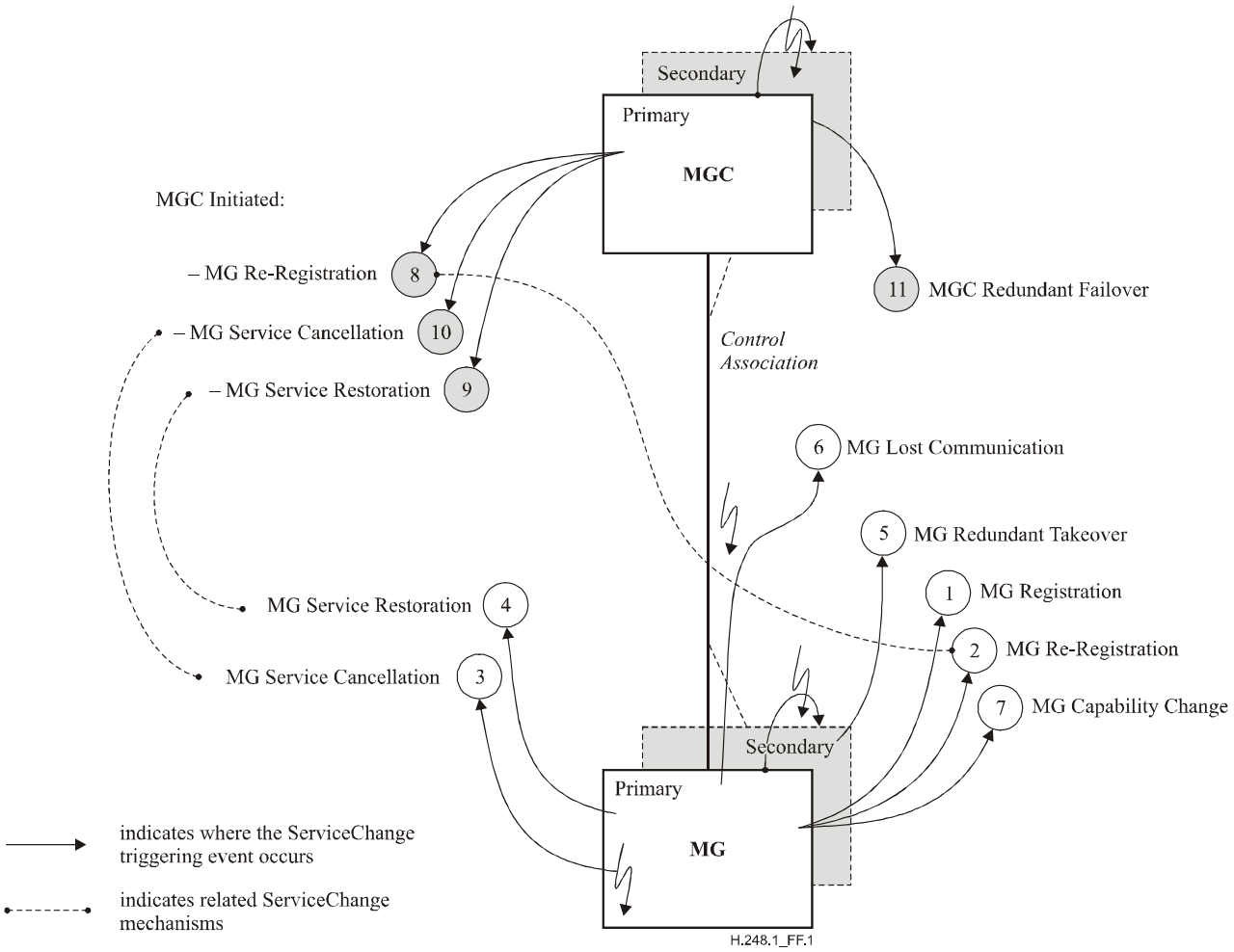

Figure F.1/H.248.1 – Trigger events for ServiceChange

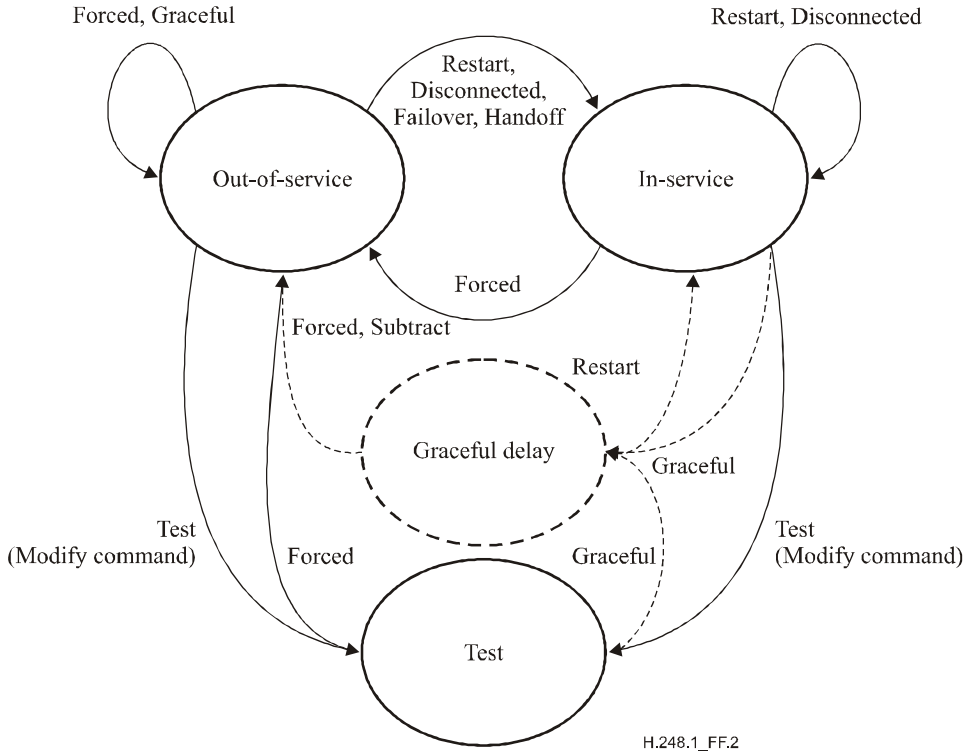

Figure F.2/H.248.1 – Simple ServiceChange state model

F.2 Control Association Definition

A control association is a communication relationship whereby an MGC is controlling an MG. A control association is instantiated via registration. The control association is terminated by the MG going OutOfService, being successfully handed off to an alternate MGC, or successfully failing over to an alternate MGC. An MG shall have at most one control association at any time except where a physical MG is split into one or more Virtual MGs. In the Virtual MG case, each Virtual MG shall have at most one control association. Only MGs may instantiate control associations. Control associations as defined here apply only within the context of media gateway control signalling. This definition should not be confused with that of control associations in the context of transport (TCP, UDP, etc.).

The use of ServiceChange Commands on the Root Termination only affects the Virtual MG to which the Root Termination belongs. Other Virtual MGs on the same physical MG are unaffected.

F.3 Events leading to ServiceChange Procedures

F.3.1 MG Registration

The MG may register in one of three ways:

- Announce its initial presence, or "registration" by using the ServiceChangeMethod "Restart" with ServiceChangeReasons 901 ("Cold Boot") or 902 ("Warm Boot"),
- Indicate that it is changing its association via the ServiceChangeMethod "Failover" and ServiceChangeReason 909 ("MGC Impending Failure") or 908 ("MG Impending Failure"),
- Indicate that it was commanded to change its association via the ServiceChangeMethod "Handoff" and ServiceChangeReason 903 ("MGC Directed Change") (F.3.11).

Registration starts when the initial ServiceChange Command is sent from the MG to the MGC and completes when the command is replied by the MGC without an alternate address or an error and the ServiceChangeProfile is agreed between the MGC and MG. The control association is

established upon completion of registration. Registration always occurs on the Root Termination. Any existing control association between the MGC and MG is terminated upon initiation of a new registration. All commands from the previous control association shall be ignored.

ServiceChangeProfile is used by registration. See F.5.5 for details.

ServiceChangeVersion is used by registration. See F.5.6 for details.

F.3.2 MG Re-Registration

MG Re-registration occurs in two cases:

1) When the MGC requests the MG to re-register (F.3.8). The MG shall then send a ServiceChange with ServiceChangeMethod "Handoff", ServiceChangeReason 903 ("MGC Directed Change") and the designated "ServiceChangeMgcID".
2) When the MGC initiates service restoration (F.3.9). The MG shall reregister by sending a ServiceChange with ServiceChangeMethod "Restart".

The re-registration is complete when the command is replied by a MGC without an alternate address or an error and the ServiceChangeProfile is agreed between the MGC and MG. If no response is received from the primary MGC then the MG shall follow the procedures of F.3.1. ServiceChangeReasons 903 ("MGC Directed Change") or 909 ("MGC Impending Failure") are recommended.

The control association is established upon completion of registration. Registration always occurs on the Root Termination. Any existing control association between the MGC and MG is terminated upon initiation of the re-registration. All commands from the previous control association shall be ignored.

ServiceChangeProfile is used by re-registration. See F.5.5 for details.

ServiceChangeVersion is used by re-registration. See F.5.6 for details.

F.3.3 MG Service Cancellation

To place the MG "OutOfService", the MG sends a ServiceChange Command with a ServiceChangeMethod of "Forced" or "Graceful" on the Root Termination. See F.4.1.1 for details.

To place a termination or group of terminations "OutOfService", the MG sends a ServiceChange with a ServiceChangeMethod of "Forced" or "Graceful" on the termination(s) in question. See F.4.1.2 and F.4.1.3 for details.

ServiceChangeDelay indicates the length of time before the service cancellation will occur. See F.5.3 for details.

To cancel a previously issued (and acknowledged) Graceful for the entire MG, the MG sends a ServiceChange Command on the Root Termination with a ServiceChangeMethod of "Restart" and ServiceChangeReason 918, "Cancel Graceful". The MG shall remain InService, and all terminations previously set OutOfService are returned to service unless otherwise signalled by the MG. In the event that a Cancel Graceful is received after the Delay timer has expired, the MG shall be deemed to have re-registered, just as if it had sent a Cold Boot.

To cancel a previously issued (and acknowledged) Graceful on a termination or group of terminations, the MG sends a ServiceChange with a ServiceChangeMethod of "Restart" and a ServiceChangeReason of "Cancel Graceful" on the termination(s) in question. The termination shall remain InService. In the event that the termination has already transitioned OutOfService, it shall be returned to service just as it would with any ServiceChange Restart.

F.3.4 MG Service Restoration

The MG Service Restoration event occurs when a MG returns to service after a failure or maintenance action.

When sent by the MG, the MG sends a ServiceChange Command to the MGC announcing that it has restarted or wishes to renegotiate the protocol version or H.248 profile. ServiceChangeMethod "Restart" and ServiceChangeReason 900 ("Service Restored") are used in this case. The ServiceChangeReason will indicate what actions may need to be taken by the MGC.

F.3.5 MG Redundant Takeover

There are two cases for MG redundant takeover:

1) Takeover initiated by Primary MG:

   When a MG is going OutOfService and wishes to relinquish processing to a designated secondary MG, it sends a ServiceChange Command to the MGC with ServiceChangeMethod "Failover" and a ServiceChangeReason of 908 ("MG Impending Failure"). The MGC will cease messaging to the primary MG and terminate the control association. The secondary MG shall then send a ServiceChange Command with ServiceChangeMethod "Restart" and a ServiceChangeReason of 900 ("Service Restored"). The MGC shall then establish a new control association with the secondary MG.

2) Takeover initiated by Secondary MG:

   When an optional secondary unit of a MG detects a maintenance or failure outage of the primary MG unit and the primary is unable to notify the MGC of the failure, the secondary MG shall send a ServiceChange Command to the MGC with ServiceChangeMethod "Failover" and a ServiceChangeReason of either 919 ("Warm Failover") or 920 ("Cold Failover"), as appropriate. The procedures of F.3.4 apply.

The MGC shall ignore all attempts at redundant takeover by unknown MGs. The MGC must have knowledge, through provisioning or other means, that the secondary MG announcing takeover is associated with the failed primary MG and therefore has authorization to takeover for the primary MG.

If the secondary MG is known and has the authority to takeover for the primary MG, the MGC shall attempt to communicate with the primary MG to determine if it still functions. This may be done via an empty AuditValue Command or other suitable method (see 11.5). If the primary MG answers, the MGC shall reject the failover attempt. If the primary MG fails to answer, the MGC shall assume that it has failed, and accept the warm or cold failover. When the MGC accepts the warm or cold failover, the control association with the primary MG is removed and a new control association with the secondary MG is established.

F.3.6 MG Lost Communication

When the MG has detected a loss and subsequent re-establishment of communication with the MGC, the MG sends a ServiceChange Command with a ServiceChangeMethod of "Disconnected" to the MGC in the current control association. If the MGC fails to respond, the MG then sends a ServiceChange Command with a ServiceChangeMethod of "Failover" and ServiceChangeReason 909 ("MGC Impending Failure") to each MGC in its list in turn until it has successfully established a new control association, or it has exhausted its list of MGCs.

If the MGC in the original control association replies to the ServiceChange Command, the control association continues without interruption, and all commands are processed as if there had been no loss of communication. Otherwise, the original control association is terminated when the MG sends a ServiceChange to a new MGC, and all commands from the previous control association are ignored. A new control association is established once the new MGC replies to the ServiceChange, completing the registration.

If the MG exhausts its list of MGCs without successfully establishing a control association, the MG waits a random amount of time and then attempts registration with the MGCs in its list again, starting with the MGC from the original control association. The MG will send a ServiceChange with a ServiceChangeMethod of "Disconnected" to the MGC from the original control association each time the MG attempts to contact it. The MG sends a ServiceChange with a ServiceChangeMethod of "Failover" to all other MGCs.

MGCs that receive ServiceChange Commands with a ServiceChangeMethod of "Disconnected" should audit the MG to determine if a state mismatch has occurred due to lost messaging.

In these scenarios, ServiceChangeReason 900 ("Service Restored") is recommended, though particular scenarios may require different ServiceChangeReason codes.

F.3.7 MG Capability Change

To announce a change in capabilities of the MG, the MG sends a ServiceChange with an appropriate ServiceChangeMethod and a ServiceChangeReason of 916 ("Packages Change") or 917 ("Capabilities Change"). The MGC should audit the MG to determine the new capabilities of the MG. For an inservice MG that sends a ServiceChange Command with ServiceChangeMethod "Restart", the control association continues uninterrupted through a capability change, and the MG is not considered to have undergone a system restart.

F.3.8 MGC Initiated MG Re-Registration

The MGC may request the MG to re-register by issuing a ServiceChange Command on Root with ServiceChangeMethod "Handoff", ServiceChangeReason 903 ("MGC Directed Change") and its own ServiceChangeMgcID (that is, that of the current MGC).

See F.3.2 for action taken by the MG.

F.3.9 MGC Initiated Service Restoration

To request that the MG restart itself, the MGC sends a ServiceChange Command to the MG with ServiceChangeMethod "Restart" and an appropriate ServiceChangeReason such as 900 ("Service Restored") or 901 ("Cold Boot"). The MG must establish a new control association in accordance with F.3.2 using the indicated ServiceChangeReason.

To restore service to a termination or group of terminations, the MGC sends a ServiceChange with ServiceChangeMethod "Restart" on the termination(s) in question. See F.4.1.2 and F.4.1.3 for actions to be taken by the MGC.

ServiceChangeDelay indicates the length of time before the service restoration will occur. See F.5.3 for details.

F.3.10 MGC Initiated Service Cancellation

F.3.10.1 Root Termination

To place the MG "OutOfService", the MGC sends a ServiceChange Command with a ServiceChangeMethod of "Forced" or "Graceful" on the Root Termination. Appropriate ServiceChangeReasons may include 905 ("Termination taken out of service"), among others. See F.4.1.1 for actions to be taken by the MGC.

ServiceChangeDelay indicates the time period at the end of which the service cancellation will occur. See F.5.3 for details.

To cancel a previously issued (and acknowledged) Graceful for the entire MG, the MGC sends a ServiceChange Command on the Root Termination with a ServiceChangeMethod of "Restart" and ServiceChangeReason 918 ("Cancel Graceful"). The MG shall remain InService, and all terminations previously set OutOfService are returned to service unless otherwise signalled by the MG. In the event that a Cancel Graceful is received after the Delay timer has expired, the MG shall

report an Error Code 502 ("Not Ready") to the MGC. The MG will be required to re-register to return to service.

F.3.10.2 Physical Terminations

To place a termination or group of terminations "OutOfService", the MGC sends a ServiceChange Command with a ServiceChangeMethod of "Forced" or "Graceful" on the termination(s) in question. Appropriate ServiceChangeReasons may include 904 ("Termination malfunctioning"), 905 ("Termination taken out of service"), 906 ("Loss of lower layer connectivity"), or 907 ("Transmission failure"), among others. See F.4.2 and F.4.3 for actions to be taken by the MGC.

ServiceChangeDelay indicates the time period at the end of which the service cancellation will occur. See F.5.3 for details.

To cancel a previously issued (and acknowledged) Graceful on a termination or group of terminations, the MG sends a ServiceChange with a ServiceChangeMethod of "Restart" and ServiceChangeReason 918 ("Cancel Graceful") on the termination(s) in question. The termination shall remain InService. In the event that the termination has already transitioned OutOfService, it shall be returned to service just as it would with any ServiceChange Restart.

F.3.10.3 Ephemeral Terminations

The MGC shall not use ServiceChange to cancel service to ephemeral terminations. Subtracting the termination from context is sufficient to eliminate the termination.

F.3.11 MGC Redundant Failover

When the MGC in a control association encounters a maintenance or failure condition such that it must go OutOfService, it may direct the MG to a specific secondary MGC by sending a ServiceChange Command with a ServiceChangeMethod of "Handoff", a ServiceChangeReason of 903 ("MGC Directed Change") and the address of the new MGC in the ServiceChangeMgcID parameter. The control association is terminated upon the receipt of the command reply from the MG. The MG then sends a ServiceChange Command to the specified MGC with a ServiceChangeMethod of "Handoff" and a ServiceChangeReason of 903 ("MGC Directed Change"). A new control association is established upon receipt of the command reply from the MGC.

In the event that the specified MGC rejects the handoff attempt or the MGC is unable to send a ServiceChange Command with ServiceChangeMethod Handoff due to failure, the MG shall revert to the procedures for lost communication outlined in F.3.6, starting with its provisioned primary.

F.4 ServiceChange Element Description

F.4.1 ServiceChangeMethod

This clause describes the behaviour of the ServiceChangeMethod on the different termination types. This clause is organized into:

- The Root Termination;
- Physical terminations;
- Ephemeral terminations.

F.4.1.1 ServiceChange Method Behaviour on the Root Termination

The ServiceChange Command on Root has different effects depending upon the ServiceChangeMethod. The results of each ServiceChangeMethod are described below:

1) Restart – Sent by the MG to announce that it has restarted, wishes to renegotiate the protocol version, profile, or is announcing a capability change. The ServiceChangeReason

indicates what actions may need to be taken by the MGC. When sent by the MGC, the MG shall restart itself using the indicated ServiceChangeReason.

2) Forced – When sent by the MG, the MG indicates that it is going OutOfService immediately. The control association is terminated by the MG upon receipt of the command reply from the MGC. When sent by the MGC, the MG shall take itself OutOfService, and terminate the control association after sending the command reply. Note that an MGC cannot bring an MG back into service, even though it may request it to go OutOfService. The MG initiates registration to the MGC to establish a new control association and indicate service restoration, just as it would for any other registration. ServiceChangeDelay has no effect on ServiceChangeMethod "Forced".

3) Graceful – When sent by the MG, it indicates that the MG is going OutOfService at the end of the ServiceChangeDelay period. The control association is terminated at the end of the ServiceChangeDelay period. When sent by the MGC, the MG shall take itself OutOfService and terminate the control association at the end of the ServiceChangeDelay period. Using a ServiceChangeDelay equal to zero or an absent ServiceChangeDelay indicates that the MG shall go OutOfService and terminate the control association when the last context is removed through subtraction of its terminations, and that the MGC shall not add new connections. The MG should set the Root Termination ServiceStates Property at the expiry of ServiceChangeDelay or the removal of all terminations from active contexts (whichever is first), to "OutOfService". To cancel a previously sent (and acknowledged) ServiceChange with ServiceChangeMethod of "Graceful", the entity initiating the ServiceChangeMethod Graceful sends a ServiceChange Command with ServiceChangeMethod Restart and the ServiceChangeReason 918 ("Cancel Graceful").

4) Failover – When sent by the MG to an MGC that is not in the current control association, the MG indicates that its current MGC has failed, and it is attempting to register with the recipient of the command. When sent by the MG to the MGC in the current control association, the primary MG indicates that a secondary MG has taken over for the failed primary. In either instance, the previous control association is terminated upon the sending of the ServiceChange Command, and the new control association is established between the MG and new MGC, or MGC and new MG upon receipt of the command reply. MGCs shall not send ServiceChange Commands with the ServiceChangeMethod "Failover".

5) Handoff – When sent by the MGC, the MGC indicates that the MG is being transferred to a new MGC. This terminates the current control association, upon receipt of the command reply. When sent from the MG to the MGC, this indicates that the MG is attempting to establish a new control association in accordance with a handoff command received from the MGC in its previous control association. When sent from the MG to the MGC, a handoff is a registration, and a new control association is established upon receipt of the command reply. The MG shall not use Handoff without being commanded to do so by an MGC.

6) Disconnected – When sent by the MG, it indicates that communication in the current control association had been lost, but is now re-established. The current control association is renewed. MGCs shall not send ServiceChange Commands with the ServiceChangeMethod "Disconnected".

F.4.1.2 ServiceChange Method Behaviour on Physical Terminations

While the MG is in an active control association, issuing the ServiceChange Command on a physical termination has different effects depending on the ServiceChangeMethod. The results of each method are described below:

1) Restart – When sent by the MG, the MG announces that the termination(s) have restarted, or is announcing a capability change. The ServiceChangeReason indicates what actions

may need to be taken by the MGC. When sent by the MGC, the MG shall restart the termination(s) using the indicated ServiceChangeReason.

2) Forced – When sent by the MG, it indicates that the termination is going OutOfService immediately. When sent by the MGC, the MG shall take the termination OutOfService immediately. In both cases, the ServiceStates parameter shall be set to "OutOfService" and the MGC is responsible for cleaning up any contexts or resources associated with the termination. ServiceChangeDelay has no effect on the ServiceChangeMethod "Forced".

3) Graceful – When sent by the MG, it indicates that the termination(s) is going OutOfService after the ServiceChangeDelay. When sent by the MGC, the MG shall take the termination(s) OutOfService at the end of the ServiceChangeDelay period. The ServiceStates Property shall be set to "OutOfService" upon expiry of the ServiceChangeDelay or when the termination(s) is removed from an active context (whichever is first) and the MGC is responsible for cleaning up any contexts or resources associated with the termination(s). Using a ServiceChangeDelay equal to zero or an absent ServiceChangeDelay indicates that the termination shall go OutOfService when it is removed from context through subtraction. The MGC shall not use the indicated termination(s) for connection until the Graceful is cancelled or the termination is brought back into service by a subsequent ServiceChange Command. To cancel a previously sent (and acknowledged) ServiceChange with ServiceChangeMethod of "Graceful", the entity initiating the Graceful sends a ServiceChange Command with ServiceChangeMethod "Restart" and the ServiceChangeReason 918 ("Cancel Graceful").

4) Failover – The ServiceChangeMethod "Failover" shall not be used with non-Root terminations.

5) Handoff – The ServiceChangeMethod "Handoff" shall not be used with non-Root terminations.

6) Disconnected – The ServiceChangeMethod "Disconnected" shall not be used with non-Root terminations.

F.4.1.3 ServiceChange Method Behaviour on Ephemeral Terminations

While the MG is in an active control association, issuing the ServiceChange Command on an ephemeral termination has different effects depending on the ServiceChangeMethod. The results of each method are described below:

1) Restart – When sent by the MG, the MG announces that the termination(s) has restarted, or is announcing a capability change. The ServiceChangeReason indicates what actions may need to be taken by the MGC. The MGC shall not send ServiceChangeMethod "Restart" for ephemeral terminations.

2) Forced – When sent by the MG, it indicates that the termination(s) is going OutOfService immediately. The MGC is responsible for subtracting the termination. The MGC shall not send ServiceChangeMethod "Forced" for ephemeral terminations. ServiceChangeDelay has no effect on ServiceChangeMethod "Forced".

3) Graceful – When sent by the MG, it indicates that the termination(s) is going OutOfService at the end of the ServiceChangeDelay period. The MGC is responsible for subtracting the termination(s) at the expiry of the ServiceChangeDelay. The MGC shall not send ServiceChangeMethod "Graceful" for ephemeral terminations. Using a ServiceChangeDelay equal to zero indicates that the termination shall be destroyed when it is removed from context through subtraction. The MG should set the termination's ServiceStates Property at the expiry of ServiceChangeDelay or the removal of the termination from an active context (whichever is first), to "Out of Service". To cancel a previously sent (and acknowledged) ServiceChange with ServiceChangeMethod of

"Graceful", the entity initiating the Graceful sends a ServiceChange Command with ServiceChangeMethod Restart and the ServiceChangeReason of 918 "Cancel Graceful".

4) Failover – The ServiceChangeMethod "Failover" shall not be used with non-Root terminations.

5) Handoff – The ServiceChangeMethod "Handoff" shall not be used with non-Root terminations.

6) Disconnected – The ServiceChangeMethod "Disconnected" shall not be used with non-Root terminations.

F.5 Use of ServiceChange parameters

F.5.1 ServiceChangeMethod

See the usage outlined in F.4.

F.5.2 ServiceChangeReason

ServiceChangeReasons allow the receiving party to alter its behaviour to fit a particular situation. For example, if a MG sent a ServiceChangeReason 901 ("Cold Boot") to the MGC when it Restarted, the MGC could assume that the MG had lost all its state and therefore it would not perform maintenance and audit tasks to assess and clean up the MG to a useable state.

Table F.1 shows with which ServiceChangeMethods particular ServiceChangeReasons may be sent.

Table F.1/H.248.1 – ServiceChangeMethod and ServiceChangeReason mapping

SC Reason	SC Method						Description
	Restart	Forced	Graceful	Disconnected	Failover	Handoff	
900	X			Root only MG only			Service Restored
901	Root only						Cold Boot
902	Root only						Warm Boot
903						Root only	MGC Directed Change
904		X	X				Term Malfunction
905		X	X				Term Taken OOS
906		X	X				Loss of lower layer connectivity
907		X	X				Transmission failure
908		Root only MG only	Root only MG only		Root only MG only		MG Impending Failure
909					Root only MG only		MGC Impending Failure
910	MG only	X	X				Media Capability Failure

Table F.1/H.248.1 – ServiceChangeMethod and ServiceChangeReason mapping

SC Reason	SC Method						Description
	Restart	Forced	Graceful	Disconnected	Failover	Handoff	
911	MG only	X	X				Modem Capability Failure
912	MG only	X	X				Mux Capability Failure
913	MG only	X	X				Signal Capability Failure
914	MG only	X	X				Event Capability Failure
915		X	X				State Loss
916	X			Root only MG only	Root only MG only		Packages Change
917	X			Root only MG only	Root only MG only		Capability Change
918	X						Cancel Graceful
919					Root only MG only		Warm Failover
920					Root only MG only		Cold Failover

F.5.3 ServiceChangeDelay

The ServiceChangeDelay is used to provide a delay before the ServiceChange Command takes effect and the ServiceStates Property of the MG or termination(s) is changed. The ServiceChange Command is replied as if execution happened upon receipt of the message, but the MG or termination(s) does not actually change state until the ServiceChangeDelay expires. A ServiceChangeDelay equal to zero is equivalent to no ServiceChangeDelay, with the exception of its combined use with the ServiceChangeMethod "Graceful". See F.4 for the effects of ServiceChangeDelay on the various ServiceChangeMethods.

F.5.4 ServiceChangeAddress

The use of the ServiceChangeAddress parameter is described in 7.2.8. The use of the ServiceChangeAddress is discouraged. If the parameter is present, it may only be used with ServiceChange Commands on the Root Termination, and any new transactions shall be sent to the new address and/or port number specified. Replies shall be sent back to the address from which the corresponding request came.

F.5.5 ServiceChangeProfile

The ServiceChangeProfile parameter allows the MGC and MG to negotiate the H.248 profile to be used in the control association. The MGC may still audit the MG to determine other supported MG capabilities. ServiceChangeProfile shall only be sent on registration or re-registration commands.

F.5.6 ServiceChangeVersion

ServiceChangeVersion is used to negotiate the H.248 protocol version to be used between the MGC and MG. ServiceChangeVersion is mandatory on initial registration, and should be sent on any other registration commands where protocol version negotiation is to take place. ServiceChangeVersion shall not be sent on non-registration commands. See F.3.1 for registration procedures.

F.5.7 ServiceChangeMgcID

The MGC may send this parameter in a ServiceChange Command directed toward the Root Termination. Upon receipt during a registration attempt, the MG shall attempt registration with the MGC at the specified address. When received in a ServiceChange Handoff Command from the MG's primary MGC, the MG shall utilize the procedures outlined in 11.5.

F.5.8 TimeStamp

The use of the optional TimeStamp parameter is described in 7.2.8. The TimeStamp parameter has no effect on the execution of ServiceChange Commands, but may be of use to the receiver of the ServiceChange Command for other purposes, such as billing or timing coordination.

F.6 ServiceChange versus TerminationState

The ServiceStates Property in the TerminationState Descriptor holds the current value of the state of the termination. ServiceStates may have three values: "InService", "OutOfService", and "Test".

The MGC shall only use Modify Commands to set the ServiceStates Property to or from Test. All other Modifies of the ServiceStates Property are a violation of protocol, and shall receive Error Code 401 ("Protocol Error"). To change the state from "InService" to "OutOfService" or vice versa, the ServiceChange Command must be used.

For terminations in the Test state, the MG shall only send ServiceChanges with a ServiceChangeMethod of "Forced" or "Graceful" to the MGC. Subsequent ServiceChange Commands may then set the termination's ServiceState Property to "InService".

Receivers should make every reasonable attempt to accept ServiceChange Commands, but in some instances rejection of the command is warranted. For example, if a MG sent a ServiceChange Command to bring a termination into service, but the MGC does not have the resources allocated to service that termination due to lack of provisioning or other reason, then the MGC may reject the command. If the receiving entity rejects the ServiceChange Command, the state of the termination remains unchanged. The sender may choose to wait some length of time and try again, or wait for the receiver to attempt to alter the termination state. Requests to take a termination OutOfService should be honoured.

Table F.2 shows which commands may be used to effect a state change on a termination.

Table F.2/H.248.1 – State transition commands and their effects

Current State	New State	Command	MGC Allowed?	MG Allowed?
InService	Test	ServiceChange	Not Possible	Not Possible
		Modify	Yes	Not Possible
InService	OutOfService	ServiceChange	Yes, physical and Root only	Yes
		Modify	No	Not Possible
OutOfService	InService	ServiceChange	Yes, but not on Root	Yes
		Modify	No	Not Possible

Table F.2/H.248.1 – State transition commands and their effects

OutOfService	Test	ServiceChange	Not Possible	Not Possible
		Modify	Yes	Not Possible
Test	InService	ServiceChange	No	No
		Modify	Yes	Not Possible
Test	OutOfService	ServiceChange	No	Yes
		Modify	Yes	Not Possible

Appendix I

Example call flows

All H.248.1 implementors must read the normative part of this Recommendation carefully before implementing from it. No one should use the examples in this appendix as stand-alone explanations of how to create protocol messages.

The examples in this appendix use SDP for encoding of the Local and Remote Descriptors. SDP is defined in RFC 2327. If there is any discrepancy between the SDP in the examples and RFC 2327, the RFC should be consulted for verification. Audio profiles used are those defined in RFC 1890, and others registered with IANA. For example, G.711 A-law is called PCMA in SDP, and is assigned profile 0. G.723.1 is called G723 and is profile 4; H.263 is called H263 and is profile 34. See also http://www.iana.org/assignments/rtp-parameters.

I.1 Residential gateway to residential gateway call

This example scenario illustrates the use of the elements of the protocol to set up a Residential Gateway to Residential Gateway call over an IP-based network. For simplicity, this example assumes that both Residential Gateways involved in the call are controlled by the same Media Gateway Controller.

I.1.1 Programming residential GW analog line terminations for idle behaviour

The following illustrates the API invocations from the Media Gateway Controller and Media Gateways to get the terminations in this scenario programmed for idle behaviour. Both the originating and terminating Media Gateways have idle analog line terminations programmed to look for call initiation events (i.e., offhook) by using the Modify Command with the appropriate parameters. The NULL Context is used to indicate that the terminations are not yet involved in a context. The Root Termination is used to indicate the entire MG instead of a termination within the MG.

In this example, MG1 has the IP address 124.124.124.222, MG2 is 125.125.125.111, and the MGC is 123.123.123.4. The Megaco port is 55555 for all three.

1) An MG registers with an MGC using the ServiceChange Command:

```
MG1 to MGC:

MEGACO/1 [124.124.124.222]
Transaction = 9998 {
    Context = - {
        ServiceChange = ROOT {Services {
```

```
            Method=Restart, Version=3,
            ServiceChangeAddress=55555, Profile=ResGW/1}
        }
    }
}
```

2) The MGC sends a reply:

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Reply = 9998 {
   Context = - {ServiceChange = ROOT {
     Services {ServiceChangeAddress=55555, Profile=ResGW/1} } }
}
```

3) The MGC programs a termination in the NULL Context. The TerminationID is A4444, the streamID is 1, the requestID in the Events Descriptor is 2222. The MID is the identifier of the sender of this message; in this case, it is the IP address and port [123.123.123.4]:55555. The Mode Property for this stream is set to SendRecv. "al" is the analog line supervision package. The Local and Remote Descriptors are assumed to be provisioned.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Transaction = 9999 {
    Context = - {
        Modify = A4444 {
            Media { Stream = 1 {
                      LocalControl {
                          Mode = SendRecv,
                          tdmc/gain=2,  ; in dB,
                          tdmc/ec=on
                      },
                  }
            },
            Events = 2222 {al/of {strict=state}}
        }
    }
}
```

The dialplan script could have been loaded into the MG previously. Its function would be to wait for the off-hook, turn on dialtone and start collecting DTMF digits. However, in this example, we use the DigitMap, which is put into place after the off-hook is detected (step 5) below).

Note that the embedded Events Descriptor could have been used to combine steps 3) and 4) with steps 8) and 9), eliminating steps 6) and 7).

4) The MG1 accepts the Modify with this reply:

```
MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Reply = 9999 {
   Context = - {Modify = A4444}
}
```

5) A similar exchange happens between MG2 and the MGC, resulting in an idle termination called A5555.

I.1.2 Collecting originator digits and initiating termination

The following builds upon the previously shown conditions. It illustrates the transactions from the Media Gateway Controller and originating Media Gateway (MG1) to get the originating termination

(A4444) through the stages of digit collection required to initiate a connection to the terminating Media Gateway (MG2).

6) MG1 detects an off-hook event from User 1 and reports it to the Media Gateway Controller via the Notify Command.

```
MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Transaction = 10000 {
   Context = - {
       Notify = A4444 {ObservedEvents =2222 {
         19990729T22000000:al/of(init=OFF)}}
   }
}
```

7) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Reply = 10000 {
    Context = - {Notify = A4444}
}
```

8) The MGC modifies the termination to play dial tone, to look for digits according to Dialplan0 and to look for the on-hook event now.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Transaction = 10001 {
    Context = - {
        Modify = A4444 {
            Events = 2223 {
                al/on(strict=state), dd/ce {DigitMap=Dialplan0}
            },
            Signals {cg/dt},
            DigitMap= Dialplan0{
(0| 00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxxx|9011x.)}
        }
    }
}
```

9) And the Modify is acknowledged.

```
MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Reply = 10001 {
    Context = - {Modify = A4444}
}
```

10) Next, digits are accumulated by MG1 as they are dialled by User 1. Dialtone is stopped upon detection of the first digit. When an appropriate match is made of collected digits against the currently programmed dialplan for A4444, another Notify is sent to the Media Gateway Controller.

```
MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Transaction = 10002 {
   Context = - {
       Notify = A4444 {ObservedEvents =2223 {
         19990729T22010001:dd/ce{ds="916135551212",Meth=UM}}}
   }
}
```

11) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Reply = 10002 {
    Context = - {Notify = A4444}
}
```

12) The controller then analyses the digits and determines that a connection needs to be made from MG1 to MG2. Both the TDM termination A4444, and an RTP termination are added to a new context in MG1. Mode is RecvOnly since Remote Descriptor values are not yet specified. Preferred codecs are in the MGC's preferred order of choice.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Transaction = 10003 {
    Context = $ {
       Add = A4444,
       Add = $ {
           Media {
             Stream = 1 {
                  LocalControl {
                      Mode = RecvOnly,

                      nt/jit=40 ; in ms
                  },
                  Local {
v=0
c=IN IP4 $
m=audio $ RTP/AVP 4
a=ptime:30
v=0
c=IN IP4 $
m=audio $ RTP/AVP 0
                  }
             }
           }
       }
    }
}
```

NOTE – The MGC states its preferred parameter values as a series of SDP blocks in Local. The MG fills in the Local Descriptor in the Reply.

13) MG1 acknowledges the new termination and fills in the Local IP address and UDP port. It also makes a choice for the codec based on the MGC preferences in Local. MG1 sets the RTP port to 2222.

```
MEGACO/3 [124.124.124.222]:55555
Reply = 10003 {
   Context = 2000 {
      Add = A4444,
      Add=A4445{
         Media {
             Stream = 1 {
                 Local {
v=0
o=- 2890844526 2890842807 IN IP4 124.124.124.222
s=-
t= 0 0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP 4
a=ptime:30
a=recvonly
```

```
                } ; RTP profile for G.723.1 is 4
             }
          }
       }
    }
}
```

14) The MGC will now associate A5555 with a new context on MG2, and establish an RTP Stream (i.e., A5556 will be assigned), SendRecv connection through to the originating user, User 1. The MGC also sets ring on A5555.

```
MGC to MG2:
MEGACO/3 [123.123.123.4]:55555
Transaction = 50003 {
    Context = $ {
       Add = A5555  { Media {
            Stream = 1 {
                 LocalControl {Mode = SendRecv} }},
          Events=1234{al/of(strict=state)},
            Signals {al/ri}
            },
       Add  = $ {Media {
            Stream = 1 {
                 LocalControl {
                    Mode = SendRecv,
                    nt/jit=40 ; in ms
                 },
                 Local {
v=0
c=IN IP4 $
m=audio $ RTP/AVP 4
a=ptime:30
                 },
                 Remote {
v=0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP 4
a=ptime:30
                 } ; RTP profile for G.723.1 is 4
             }
          }
       }
    }
}
```

15) This is acknowledged. The stream port number is different from the control port number. In this case it is 1111 (in SDP).

```
MG2 to MGC:
MEGACO/3 [125.125.125.111]:55555
Reply = 50003 {
   Context = 5000 {
     Add = A5555,
       Add = A5556{
          Media {
             Stream = 1 {
                 Local {
v=0

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP 4
```

```
}
            } ; RTP profile for G.723.1 is 4
          }
        }
   }
}
```

16) The above IPAddr and UDPport need to be given to MG1 now.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Transaction = 10005 {
  Context = 2000 {
    Modify = A4444 {
      Signals {cg/rt}
    },
    Modify = A4445 {
       Media {
            Stream = 1 {
                Remote {
v=0

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP 4
                }
            } ; RTP profile for G.723.1 is 4
        }
    }
  }
}

MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Reply = 10005 {
   Context = 2000 {Modify = A4444, Modify = A4445}
}
```

17) The two gateways are now connected and User 1 hears the ringback. The MG2 now waits until User2 picks up the receiver and then the two-way call is established.

```
From MG2 to MGC:

MEGACO/3 [125.125.125.111]:55555
Transaction = 50005 {
   Context = 5000 {
       Notify = A5555 {ObservedEvents =1234 {
         19990729T22020002:al/of(init=off)}}
   }
}

From MGC to MG2:

MEGACO/3 [123.123.123.4]:55555
Reply = 50005 {
    Context = - {Notify = A5555}
}
```

```
From MGC to MG2:

MEGACO/3 [123.123.123.4]:55555
Transaction = 50006 {
   Context = 5000 {
      Modify = A5555 {
         Events = 1235 {al/on(strict=state)},
         Signals ; to turn off ringing
      }
   }
}

From MG2 to MGC:

MEGACO/3 [125.125.125.111]:55555
Reply = 50006 {
 Context = 5000 {Modify = A4445}
}
```

18) Change mode on MG1 to SendRecv, and stop the ringback.

```
MGC to MG1:
MEGACO/3 [123.123.123.4]:55555
Transaction = 10006 {
   Context = 2000 {
      Modify = A4445 {
         Media {
            Stream = 1 {
               LocalControl {
                  Mode=SendRecv
               }
            }
         }
      },
      Modify = A4444 {
         Signals
      }
   }
}

from MG1 to MGC:
MEGACO/3 [124.124.124.222]:55555
Reply = 10006 {
   Context = 2000 {Modify = A4445, Modify = A4444}}
```

19) The MGC decides to Audit the RTP termination on MG2.

```
MEGACO/3 [123.123.123.4]:55555
Transaction = 50007 {
   Context = 5000 {AuditValue = A5556{
      Audit{Media, DigitMap, Events, Signals, Packages, Statistics }}
   }
}
```

20) The MG2 replies.

```
MEGACO/3 [125.125.125.111]:55555
Reply = 50007 {
   Context = 5000 {
AuditValue = A5556 {
          Media {
             TerminationState { ServiceStates = InService,
                 Buffer = OFF },
             Stream = 1 {
                 LocalControl { Mode = SendRecv,
```

```
                   nt/jit=40 },
                 Local {
v=0

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP  4
a=ptime:30
                },
                 Remote {
v=0

o=- 2890844526 2890842807 IN IP4 124.124.124.222
s=-
t= 0 0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP  4
a=ptime:30
                 } } },
          Events,
          Signals,
          DigitMap,
          Packages {nt-1, rtp-1},
          Statistics { rtp/ps=1200,  ; packets sent
                       nt/os=62300, ; octets sent
                       rtp/pr=700, ; packets received
                       nt/or=45100, ; octets received
                       rtp/pl=0.2,  ; % packet loss
                       rtp/jit=20,
                       rtp/delay=40 } ; avg latency
       }
    }
}
```

21) When the MGC receives an on-hook signal from one of the MGs, it brings down the call. In this example, the user at MG2 hangs up first.

```
From MG2 to MGC:

MEGACO/3 [125.125.125.111]:55555
Transaction = 50008 {
   Context = 5000 {
       Notify = A5555 {ObservedEvents =1235 {
          19990729T24020002:al/on(init=off)}
       }
   }
}

From MGC to MG2:

MEGACO/3 [123.123.123.4]:55555
Reply = 50008 {
    Context = - {Notify = A5555}
}
```

22) The MGC now sends both MGs a Subtract to take down the call. Only the Subtract Commands to MG2 are shown here. Each termination has its own set of statistics that it gathers. An MGC may not need to request both to be returned. A5555 is a physical termination, and A5556 is an RTP termination.

```
From MGC to MG2:

MEGACO/3 [123.123.123.4]:55555
Transaction = 50009 {
   Context = 5000 {
      Subtract = A5555 {Audit{Statistics}},
      Subtract = A5556 {Audit{Statistics}}
   }
}

From MG2 to MGC:

MEGACO/3 [125.125.125.111]:55555
Reply = 50009 {
   Context = 5000 {
     Subtract = A5555 {
          Statistics {
             nt/os=45123, ; Octets Sent
             nt/or=45123, ; Octets Received
             nt/dur=40000 ; in milliseconds
             }
       },
       Subtract = A5556 {
          Statistics {
             rtp/ps=1245,  ; packets sent
             nt/os=62345,  ; octets sent
             rtp/pr=780,   ; packets received
             nt/or=45123,  ; octets received
             rtp/pl=10,    ; % packets lost
             rtp/jit=27,
             rtp/delay=48, ; average latency
             nt/dur=38000  ; in millisec
          }
       }
   }
}
```

23) The MGC now sets up both MG1 and MG2 to be ready to detect the next off-hook event. (See step 1). Note that this could be the default state of a termination in the NULL context, and if this were the case, no message need be sent from the MGC to the MG. Once a termination returns to the NULL context, it goes back to the default termination values for that termination.

Appendix II

H.248 Package template

New and updated H.248 packages should be defined using the following template. It is in an ITU Recommendation format. Editors from non-ITU organizations should, at a minimum, use the structures in clause 6 of the template. The H.248 packages keywords are indicated in bold. Sections to be filled in are indicated by "< >". For detailed information on how to create a new package, see clause 12.

NOTE – Package specific error codes require IANA registration. Error Code assignment shall follow the same rules as described in 4.1/H.248.8, "*Assigning Error Codes*".

ITU-T Recommendation H.248.<xxx>

Gateway Control Protocol: <xxx> Packages

1 Scope

<The scope of the package>

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation

– ITU-T Recommendation H.248.1 (<xx/xxxx>), *Gateway Control Protocol: Version< x>*

– <Other references>

3 Terms and definitions

This Recommendation uses the following terms and definitions:

<terms and definitions>

4 Abbreviations

This Recommendation uses the following abbreviations:

<abbreviations>

5 Conventions

<conventions>

6 <Package Title>

Package Name: <name>

PackageID: <PackageID "text", binary <0x????> >

Description: <description>

Version: <version>

Extends: <extended PackageID and version>

6.1 Properties

6.1.1 <Property Title>

Property Name: <name>

PropertyID: <text ID, binary ID (0x????)>

Description: <description>

Type: <type>

Possible values: <values>

Default: <value>

Defined in: <Local, Remote, LocalControl, TerminationState, ContextAttribute>

Characteristics: <ReadOnly, Read/Write>

6.2 Events

6.2.1 <Event Title>

Event Name: <event name>

EventID: <(text ID), (binary ID (0x????)>

Description: <description>

6.2.1.1 EventsDescriptor parameters

6.2.1.1.1 <Parameter Title>

Parameter Name: <name>

ParameterID: <(text ID), (binary ID (0x????)>

Description: <description>

Type: <types>

Optional: <yes/no>

Possible values: <values>

Default: <value>

6.2.1.2 ObservedEventsDescriptor parameters

6.2.1.2.1 <Parameter Title>

Parameter Name: <name>

ParameterID: <(text ID), (binary ID (0x????)>

Description: <description>

Type: <types>

Optional: <yes/no>

Possible values: <values>

Default: <value>

6.3 Signals

6.3.1 <Signal Title>

Signal Name: <name>

SignalID: <(text ID), (binary ID (0x????)>

Description: <description>

Signal Type: <type>

Duration: <duration>

6.3.1.1 Additional parameters

6.3.1.1.1 <Parameter Title>

Parameter Name: <name>

ParameterID: <(text ID), (binary ID (0x????)>

Description: <description>

Type: <types>

Optional: <yes/no>

Possible values: <values>

Default: <value>

6.4 Statistics

6.4.1 <Statistic Title>

Statistic Name: <name>

Statistic ID: <(text ID), (binary ID (0x????)>

Description: <description>

Type: <type>

Possible values: <values>

Level: <Termination, Stream, Either>

6.5 Error Codes

6.5.1 <Error Code Title>

Error Code #: <number>

Name: <name>

Definition: <definition>

Error Text in the Error Descriptor: <error text to return>

Comment: <comment>

6.6 Procedures

<The procedures associated with the package>

Appendix III

H.248 Profile Definition template

New H.248 profiles should be defined using the following profile template. It is in an ITU Recommendation format. Editors from non-ITU organization should, at a minimum, use the structures in clause 6 below. If this template format is not used, then editors of profiles should ensure that the headings and points in this appendix are covered by their profile.

The headings in the structure below represent items that may be considered as optional. For profile definitions, a certain H.248.1 item may be unused despite being mandatory in H.248.1. These items are also included in the structure below. Non-listed items are to be considered mandatory by the H.248.1 protocol. In the profile template below, elements can be defined as "optional" and "mandatory". "Optional" means that it is optional for either the sender or the receiver to include the element in a message. If the receiving entity receives an optional element that it has not implemented as per 6.2.3 it should send Error Code 501 ("Not Implemented").

The editor should provide written description in each of the subclauses below if it furthers clarifies H.248.1 behaviour. For example, if the Move Command (see 6.8.4) is limited to certain termination types, this should be indicated.

Italics text is to be removed.

<text> in brackets is to be filled in.

ITU-T Recommendation <xxx>

<Profile Title>

1 Scope

<The scope of the profile>

2 References

<References>

3 Terms and definitions

This Recommendation uses the following terms and definitions:

<terms and definitions>

4 Abbreviations

This Recommendation uses the following abbreviations:

<abbreviations>

5 Conventions

<conventions>

6 Profile description

6.1 Profile identification

Profile name:	<name 1-64 characters>
Version:	<version 1-99>

The name and version of the profile that is sent in the ServiceChange Command.

6.2 Summary

<Description>

A description of what the profile is.

6.3 Gateway Control Protocol version

<Version number>

The minimum H.248 version required to support the profile. This should be based upon base syntax support and not an arbitrary version assignment. This is related to the ServiceChangeVersion in 6.8.8.

6.4 Connection model

<Describe in words and diagrams>

A description of the allowed termination configurations in a context.

Maximum number of contexts:	<Integer>
Maximum number of terminations per context:	<Integer>
Allowed termination type combinations in a context:	<e.g., Context[a](IP,TDM), Context[b](TDM,AAL 2), etc.>

6.5 Context attributes

Context attribute	Supported	Values supported
Topology	<Yes/No>	See 6.7.8
Priority Indicator	<Yes/No>	<1-15>
Emergency Indicator	<Yes/No>	NA
IEPS Indicator	<Yes/No>	NA
ContextAttribute Descriptor	<Yes/No>	If "Yes", see 6.8.9 for details of supported attributes.
ContextIDList Parameter	<Yes/No>	NA

Is the AND/OR Select operation Context Attribute supported?

AND/OR Context Attribute	<Yes/No>	<AND/OR/BOTH>

6.6 Terminations

6.6.1 Termination names

<The TerminationID structure>

Identify the termination identities associated with physical, ephemeral and multiplex terminations.

6.6.2 Multiplexed terminations

Multiplex terminations supported?	<Yes/No>

If yes, then:

Multiplex types supported:	< H.221, H.223, H.226, V.76, N × 64K>
Maximum number of terminations connected to multiplex:	<Integer>

Are multiplexed terminations used? If so, describe.

6.7 Descriptors

6.7.1 Stream Descriptor

Maximum number of streams per termination type	<TerminationType>	<Integer>

If more than 1:

Stream configuration:	<Describe the allowed configurations. Is more than one audio stream allowed? Etc.>

6.7.1.1 LocalControl Descriptor

Are the ReserveGroup and ReserveValue properties used?

If not generic, list appropriate termination and stream types.		**Termination type**	**Stream type**
ReserveGroup used:	<Yes/No>	<Type>	<Type>
ReserveValue used:	<Yes/No>	<Type>	<Type>

Which StreamMode values are used?

Termination type	**Stream type**	**Allowed StreamMode Values**
<Type>	<Type>	<SendOnly, RecvOnly, SendRecv, Loopback>

6.7.2 Events Descriptor

All events contained in this profile may be set on any termination/stream [with the following exceptions].

NOTE – The text in [] is optional and only included if there are such exceptions.

Events settable on termination types and stream types:	<Yes/No>		
If yes	**EventID**	**Termination type**	**Stream type**
	<Event name and Identity e.g., Generic Error Event (g/cause, 0x0001/0x0001)>	<Type>	<Stream type, e.g., Audio/Video/Data or StreamID>

Is EventBuffer Control used?

EventBuffer Control used:	<Yes/No>

Is event KeepActive used?

KeepActive used on events:	<Yes/No>

Is embedding in Events used?

Embedded events in an Events Descriptor:	<Yes/No>
Embedded signals in an Events Descriptor:	<Yes/No>

Are Regulated EmbeddedEvents Supported?

Regulated Embedded events are triggered on:	<None / Specify individual event>

Is the ResetEventsDescriptor Flag used?

ResetEventsDescriptor used with events:	<ALL / None / Specify individual event>

What Notification Behaviour is supported?

NotifyImmediate:	<ALL Events / None / Specify individual events>
NotifyRegulated:	<ALL Events / None / Specify individual events>
NeverNotify:	<ALL Events / None / Specify individual events>

6.7.3 EventBuffer Descriptor

Is this supported?

EventBuffer Descriptor used:	<Yes/No>	
If yes	**EventIDs**	<Event name and Identity e.g., Generic Error Event (g/cause, 0x0001/0x0001) or ALL>

6.7.4 Signals Descriptor

All signals contained in this profile may be set on any termination/stream [with the following exceptions].

NOTE – The text in [] is optional and only included if there are such exceptions.

The setting of signals is dependant on termination or streams types:	<Yes/No> NOTE – "No" means that all signals can be played on any termination or stream. If "Yes", any signal not listed below may be played on any termination or stream.		
If yes	**SignalID**	**Termination Type**	**Stream Type / ID**
	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001)>	<Type>	<Stream Type e.g., Audio/Video/Data or StreamID>

Are signal lists supported? If supported, what is the maximum number of signals per list per termination/stream type supporting lists.

Signals Lists supported:	<Yes/No>	
If yes	**Termination Type Supporting Lists:**	<Type/ALL>
	Stream Type Supporting lists:	<Type/ALL>
	Maximum number of signals to a signal list:	<Integer>
	Intersignal delay parameter supported:	<Yes/No>

Is overriding signal type and duration supported?

Signal type and duration supported:	<Yes/No>	
If yes	**SignalID**	**Type or duration override**
	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001) or "ALL">	<Type, Duration, Both>

Is Signal Direction supported?

Signal Direction supported:	<Yes/No>

Is "notifyCompletion" supported? What types are supported? Is the RequestID used with "NotifyCompletion"?

NotifyCompletion supported:	<Yes/No>	
If yes	**SignalID**	**Type of completion supported**
	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001) or ALL>	<ALL, TO, EV, ED, NC, PI>
RequestID Parameter supported:	<Yes/No>	

Can multiple signals be played simultaneously?

Signals played simultaneously:	<Yes/No>	
If yes	**SignalIDs that can be played simultaneously:**	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001) or ALL>

Is "KeepActive" supported for signals?

KeepActive used on signals:	<Yes/No>

6.7.5 DigitMap Descriptor

Are DigitMaps supported? If so describe the names, structures and timers.

DigitMaps supported:	<Yes/No>		
If yes	**DigitMap Name**	**Structure**	**Timers**
	<name>	<Describe>	<timers>

6.7.6 Statistics Descriptor

Are statistics supported on terminations, streams or both?

Statistics supported on:	<Termination / Stream / Both>

Are statistics to be reported?

Statistics reported on Subtract:	<Yes/No>	
If yes	**StatisticIDs reported:**	<Statistics name and Identity e.g., Packets Sent (rtp/ps, 0x000c/0x0004) or ALL>

6.7.7 ObservedEvents Descriptor

Is the detection time supported?

Event detection time supported:	<Yes/No>

6.7.8 Topology Descriptor

If used what are the allowed settings?

Allowed triples:	<(T1, T2, oneway) etc.>

6.7.9 Error Descriptor

Which H.248.8 and package defined error codes are supported?

Error Codes sent by the MGC:

Supported H.248.8 Error Codes:	<ALL H.248.8, list of individual numbers>
Supported Error Codes defined in packages:	For a list of error codes see 6.14.x <Reference to the appropriate subclause in 6.14 below>

Error Codes sent by the MG:

Supported H.248.8 Error Codes:	<ALL H.248.8, list of individual numbers>
Supported Error Codes defined in packages:	For a list of error codes see 6.14.x <Reference to the appropriate subclause in 6.14 below>

6.8 Command API

NOTE – It is assumed that an Error Descriptor may be returned in any command reply.

6.8.1 Add

What descriptors can be used in an Add request?

Descriptors used by Add request:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit>

What descriptors can be used in an Add reply?

Descriptors used by Add reply:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit, Statistics>

6.8.2 Modify

What descriptors can be used in a Modify request?

Descriptors used by Modify request:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit>

What descriptors can be used in a Modify reply?

Descriptors used by Modify reply:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit>

6.8.3 Subtract

Can an Audit Descriptor be used in a Subtract request?

Descriptors used by Subtract request:	<Audit>

Can a Statistics Descriptor be used in a Subtract reply?

Descriptors used by Subtract reply:	<Statistics>

6.8.4 Move

Is the Move command used? Some context configurations may not use this.

Move command used:	<Yes/No>

If used:

Descriptors used by Move request:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit, Statistics>
Descriptors used by Move reply:	<Media, Mux, Events, EventBuffer, Signals, DigitMap, Audit, Statistics>

6.8.5 AuditValue

Which descriptors and/or individual properties, signal, events or statistics can be audited?

Audited Properties:	<Property name and Identity e.g., Maximum number of contexts (Root/maxNumberOfContexts, 0x0002/0x0001), ALL or None>	<Descriptor: Local, Remote, LocalControl, TerminationState>
Audited Statistics:	<Statistics name and Identity e.g., Packets Sent (rtp/ps, 0x000c/0x0004), ALL or None>	
Audited Signals:	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001), ALL or None>	
Audited Events:	<Event name and Identity e.g., Generic Error Event (g/cause, 0x0001/0x0001), ALL or None>	
Packages Audit possible:	*Can the Package Descriptor be audited?* <Yes/No>	

6.8.6 AuditCapabilities

Which descriptors and/or individual properties, signal, events or statistics can be audited?

Audited Properties:	<Property name and Identity e.g., Maximum number of contexts (Root/maxNumberOfContexts, 0x0002/0x0001), ALL or None>	<Descriptor: Local, Remote, LocalControl, TerminationState>
Audited Statistics:	<Statistics name and Identity e.g., Packets Sent (rtp/ps, 0x000c/0x0004), ALL or None>	
Audited Signals:	<Signal name and Identity e.g., Playtone (tonegen/pt, 0x0003/0x0001), ALL or None>	
Audited Events:	<Event name and Identity e.g., Generic Error Event (g/cause, 0x0001/0x0001), ALL or None>	

Is scoped auditing possible?

Audited Properties / ContextAttributes used for a scoped audit:	<None / ALL / Specify individual>

6.8.7 Notify

Which descriptors can be used in a Notify Command?

Descriptors used by Notify Request or Reply:	<ObservedEvents, Error>

6.8.8 ServiceChange

Which ServiceChangeMethods and Reasons are supported?

ServiceChangeMethods and ServiceChangeReasons sent by MGC:

ServiceChangeMethods supported:	ServiceChangeReasons supported:
<Graceful, Forced, Restart, Handoff, Failover, ALL, Other?>	< 900-920 >

ServiceChangeMethods and ServiceChangeReasons sent by MG:

ServiceChangeMethods supported:	ServiceChangeReasons supported:
<Graceful, Forced, Restart, Disconnected, Handoff, Failover, ALL, Other?>	< 900-920 >

Is ServiceChangeAddress used?

ServiceChangeAddress used:	<Yes/No>

Is ServiceChangeDelay used?

ServiceChangeDelay used:	<Yes/No>	
If yes	**Valid time period:**	<0-x> ms

Is ServiceChangeIncomplete Flag used?

ServiceChange Incomplete Flag used:	<Yes/No>

Which version of ITU-T Rec. H.248.1 is used by ServiceChangeVersion? The lowest value here should be the minimum version defined in 6.3.

Version used in ServiceChangeVersion:	<1, 2, 3>

Can multiple profiles be supported according to ITU-T Rec. H.248.18?

Profile negotiation as per ITU-T Rec. H.248.18:	<Yes/No>

6.8.9 Manipulating and auditing context attributes

Which Context Attributes may be manipulated and/or audited?

Context Attributes manipulated:	<Topology, Emergency, Priority, IEPS Indicator, ContextAttribute Descriptor (list attribute names), ALL>
Context Attributes audited:	<Topology, Emergency, Priority, IEPS Indicator, ContextAttribute Descriptor (list attribute names), ALL>

6.9 Generic command syntax and encoding

Specifies what encodings are supported by the profile.

Supported encodings:	<Text and Binary, Binary, Text>

6.10 Transactions

What is the maximum number of TransactionRequest/TransactionReplies per message?

Maximum number of TransactionRequests / TransactionReplies / TransResponseAcks / Segment Replies per message:	<Integer>

What is the maximum number of commands per TransactionRequest?

Maximum number of commands per TransactionRequest:	<Integer>

What is the maximum number of commands per TransactionReply?

Maximum number of commands per TransactionReply:	<Integer>

Can commands be marked "Optional"? Describe.

Commands able to be marked "Optional":	<Add, Modify, Move, Subtract, Auditvalue, Auditcapability, Servicechange, All, None>

Specify the values of the transaction timers

Transaction timer:	Value
normalMGExecutionTime	<Integer or "Provisioned">
normalMGCExecutionTime	<Integer or "Provisioned">
MGOriginatedPendingLimit	<Integer or "Provisioned">
MGCOriginatedPendingLimit	<Integer or "Provisioned">
MGProvisionalResponseTimerValue	<Integer or "Provisioned">
MGCProvisionalResponseTimerValue	<Integer or "Provisioned">

6.11 Messages

MGC/MG Naming Conventions: MID addressing associated with the names of the MGC/MG.

<Describe>

Indicate the maximum number of transactions per message (this table may be omitted if not significant)

Maximum number of transactions per message:	<Integer >

6.12 Transport

Specifies what H.248 subseries transports are supported by the profile.

Supported transports:	<UDP, TCP, SCTP, MTP3B, SSCOP/AAL 5, ALF/AAL 5>

Is segmentation supported and if so by which method?

Segmentation supported:	<No, Inherent in Transport, H.248 Segmentation>

Is control association monitoring (see 11.6) used and if so by which method?

Control Association Monitoring supported:	<No, Inherent in Transport, Empty AuditValue on Root, H.248.14>

6.13 Security

Specifies the security mechanisms used.

Supported security:	<None, Interim AH Scheme, IPSec>

6.14 Packages

Specifies the packages that are supported in this profile.

Mandatory: specifies the packages that shall be supported in this profile.

Mandatory packages:		
Package name	**PackageID**	**Version**
<name>	<xxxx, (0x00xx)>	<1, 2, 3, …>

Optional: specifies the packages that may be supported in the profile.

Optional packages:			
Package name	**PackageID**	**Version**	**Support dependent on:**
<name>	<xxx, 0x00??>	<1, 2, 3, …>	<Describe>

Package usage information

This table specifies how the packages above will be used. For example:

- *it lists whether the properties/signals/events/statistics are optional or mandatory;*
- *if the value of the property/signal/event provisioned should be specified (e.g., names and number of cycles for an H.248.7 announcement);*

Specifies the values of properties which are specified as provisioned.

Package Usage Information

6.14.x <Package Name>

Properties	**Mandatory/Optional**	**Used in command:**	**Supported values:**	**Provisioned value:**
<name and Identity e.g., Packets Sent (rtp/ps, 0x000c/0x0004), ALL or None>	<M/O>	<ADD, MOD, MOVE, AUDITVALUE, AUDITCAP>	<Values / ALL >	<Value / Not Applicable>
Signals	**Mandatory/Optional**	**Used in command:**		**Duration provisioned value:**
<name and Identity >	<M/O>	<ADD, MOD, MOVE, AUDITVALUE, AUDITCAP>		<Value / Not Applicable>
	Signal parameters	**Mandatory/Optional**	**Supported values:**	**Duration provisioned value:**
	<name and Identity>	<M/O>	<Values / ALL>	<Value / Not Applicable>
Events	**Mandatory/Optional**	**Used in command:**		
<name and Identity >	<M/O>	<ADD, MOD, MOVE, NOTIFY, AUDITVALUE, AUDITCAP>		
	Event parameters	**Mandatory/Optional**	**Supported values:**	**Provisioned value:**
	<name and Identity>	<M/O>	<Values / ALL>	<Value / Not Applicable>
	ObservedEvent parameters	**Mandatory/Optional**	**Supported values:**	**Provisioned value:**
	<name and Identity>	<M/O>	<Values / ALL>	<Value / Not Applicable>
Statistics	**Mandatory/Optional**	**Used in command:**	**Supported values:**	
<name and Identity >	<M/O>	<ADD, MOD, MOVE, SUBTRACT, AUDITVALUE, AUDITCAP>	<Values / ALL >	
Error Codes	**Mandatory/Optional**			
<number>	<M/O>			

Additional restrictions may be tabulated as the user desires.

6.15 Mandatory support of SDP and Annex C information elements

Specifies what SDP attributes and Annex C information elements are to be supported.

Supported Annex C and SDP information elements:		
Information Element	**Annex C Support**	**SDP Support**
<name>	<Annex C property>	<Describe>

6.16 Optional support of SDP and Annex C information elements

Specifies what SDP attributes and Annex C information elements may be supported.

Optional Annex C and SDP information elements:			
Information Element	**Annex C Support**	**SDP Support**	**Support Dependent on:**
<name>	<Annex C property>	<Describe>	<Describe>

6.17 Procedures

Specifies the procedures that are associated with the profile.

It is recommended that the procedures take the following format:

6.17.1 <Procedure name>

When the procedure <name> is required, the following is initiated.

An <ADD.req, MOD.req, MOV.req, SUB.req, AuditValue.req, AuditCapability.req, ServiceChange.req, Notify.req> command is sent from the <MGC/MG> to the <MG/MGC> with the following information.

< Insert information in an appropriate format. For example, the descriptors, property, signal, event, statistic names or an abstraction of the actual parameters in terms of information elements.

Upon reception of the command, the <MG/MGC> shall:

- <Indicate actions>

Upon completion of processing command (1) an <ADD.reply, MOD.reply, MOV.reply, SUB.reply, AuditValue.reply, AuditCapability.reply, ServiceChange.reply, Notify.reply> command is sent from the <MG/MGC> to the <MGC/MG> with the following information.

< Insert information in an appropriate format. For example, the descriptors, property, signal, event, statistic names or an abstraction of the actual parameters in terms of information elements.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	**Audiovisual and multimedia systems**
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2006