INTERNATIONAL TELECOMMUNICATION UNION

ITU-T H.248.1

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(03/2002)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Communication procedures

Gateway control protocol: Version 1

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.399
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

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

ITU-T Recommendation H.248.1

Gateway control protocol: Version 1

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 ITU-T Rec. H.323.

NOTE – This Recommendation comprises the renumbering of ITU-T Rec. H.248, its Annexes A through E, and its Appendix I.

Source

ITU-T Recommendation H.248.1 was prepared by ITU-T Study Group 16 (2001-2004) and approved under the WTSA Resolution 1 procedure on 29 March 2002.

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.

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 1

1 Scope

This Recommendation defines the protocols used between elements of a physically decomposed multimedia gateway, used in accordance with the architecture as specified in ITU-T Rec. H.323. There are no functional differences from a system view between a decomposed gateway, with distributed subcomponents 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.

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.

2.1 Normative references

- ITU-T Recommendation H.225.0 (2000), *Call signalling protocols and media stream packetization for packet-based multimedia communication systems*.
- ITU-T Recommendation H.235 (2000), *Security and encryption for H-Series (H.323 and other H.245-based) multimedia terminals*.
- ITU-T Recommendation H.245 (2001), *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.248.8 (2002), *Gateway control protocol: Error code and service change reason description*.
- ITU-T Recommendation H.323 (2000), *Packet-based multimedia communication 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*.
- ITU-T Recommendation I.371 (2000), *Traffic control and congestion control in B-ISDN*.
- ITU-T Recommendation Q.763 (1999), *Signalling System No. 7 – ISDN user part formats and codes*.
- ITU-T Recommendation Q.765.5 (2000), *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*.
- 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*.
- ITU-T Recommendation Q.2941.1 (1997), *Digital Subscriber Signalling System No. 2 – Generic identifier transport*.
- ITU-T Recommendation Q.2961.1 (1995), *Additional signalling capabilities to support traffic parameters for the tagging option and the sustainable cell rate parameter set*.
- ITU-T Recommendation Q.2961.2 (1997), *Support of ATM transfer capability in the broadband bearer capability information element*.
- ITU-T Recommendation Q.2965.1 (1999), *Digital subscriber signalling system No. 2 – Support of Quality of Service classes*.
- 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*.
- ITU-T Recommendation X.213 (2001), *Information technology – Open Systems Interconnection – Network service definition plus Amendment 1 (1997), Addition of the Internet protocol address format identifier*.
- ITU-T Recommendation X.680 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- ITU-T Recommendation X.690 (2002), *Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.
- ATM Forum (1996), *ATM User-Network Interface (UNI) Signalling Specification – Version 4.0*.
- IETF RFC 1006, *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 (1999), *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 (1988), *Telephone user part signalling procedures*.
- ITU-T Recommendation Q.764 (1999), *Signalling system No. 7 – ISDN user part signalling procedures*.
- ITU-T Recommendation Q.1902.4 (2001), *Bearer independent call control protocol (Capability Set 2): Basic call procedures*.
- 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 1889 (1996), *RTP: A Transport Protocol for Real-Time Applications*.
- IETF RFC 1890 (1996), *RTP Profile for Audio and Video Conferences with Minimal Control*.
- IETF RFC 2401 (1998), *Security Architecture for the Internet Protocol*.
- IETF RFC 2460 (1998), *Internet Protocol, Version 6 (IPv6) Specification*.
- IETF RFC 2543 (1999), *SIP: Session Initiation Protocol*.
- IETF RFC 2805 (2000), *Media Gateway Control Protocol Architecture and Requirements*.

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 defines the following terms:

ALF	Application Level Framing
ATM	Asynchronous Transfer Mode
CAS	Channel Associated Signalling
DTMF	Dual Tone Multi-Frequency
FAS	Facility Associated Signalling
GSM	Global System for Mobile communications
GW	Gateway
IANA	Internet Assigned Numbers Authority (superseded by Internet Corporation for Assigned Names and Numbers – ICANN)
IP	Internet Protocol
ISUP	ISDN User Part
IVR	Interactive Voice Response
MG	Media Gateway
MGC	Media Gateway Controller
NFAS	Non-Facility Associated Signalling
PRI	Primary Rate Interface
PSTN	Public Switched Telephone Network

QoS Quality of Service

RTP Real-time Transport Protocol

SCN Switched Circuit Network

SG Signalling Gateway

SS7 Signalling System No. 7

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 modem, and 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 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.

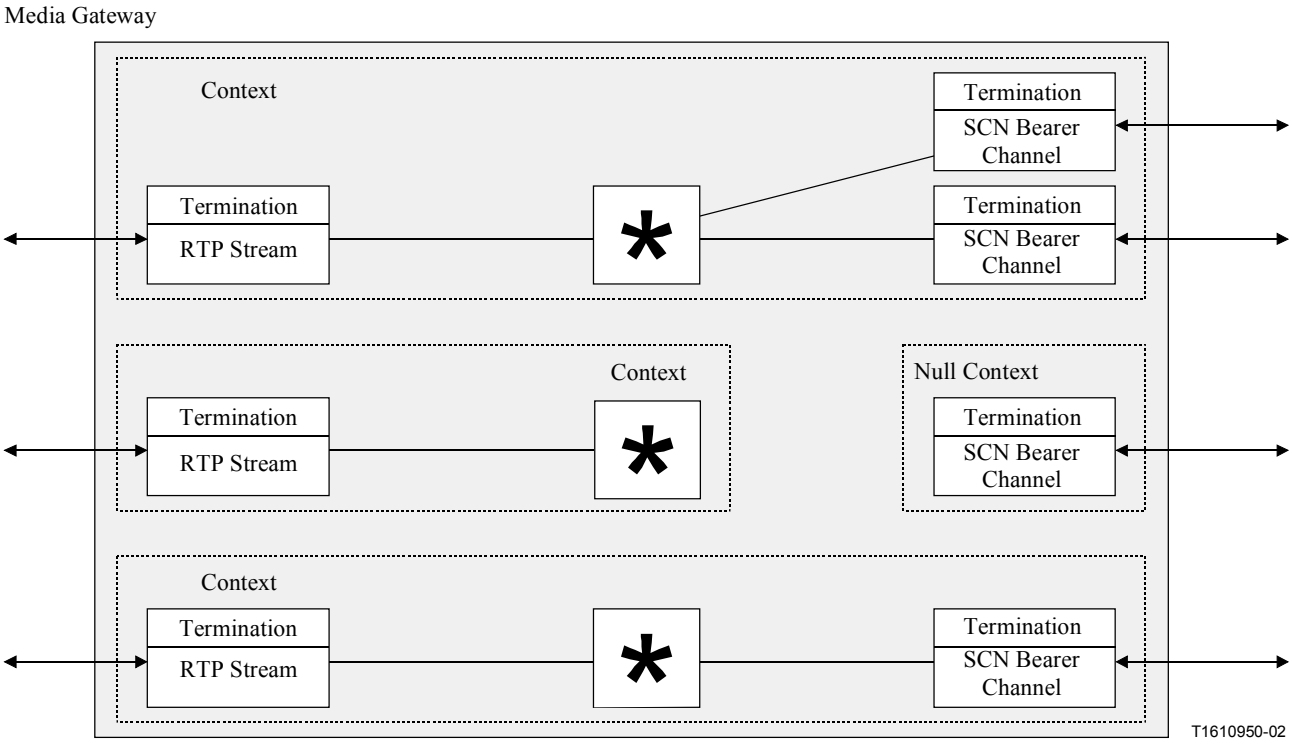

Figure 1/H.248.1 – Example of H.248.1 Connection Model

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.

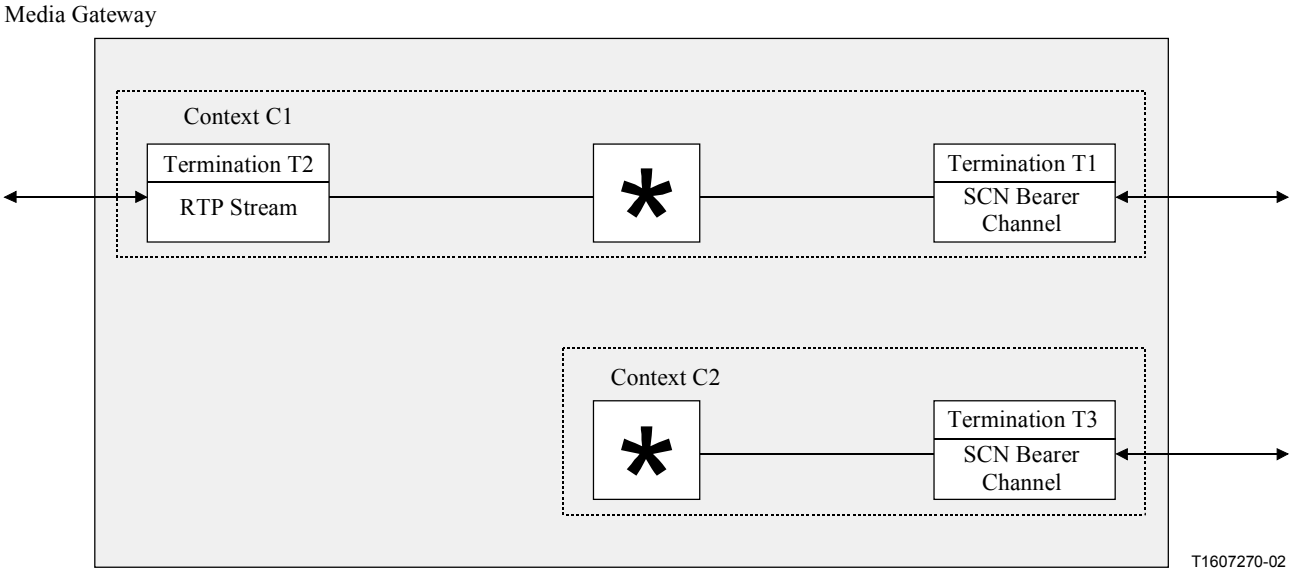

Figure 2/H.248.1 – Example Call Waiting Scenario/Alerting Applied to T1

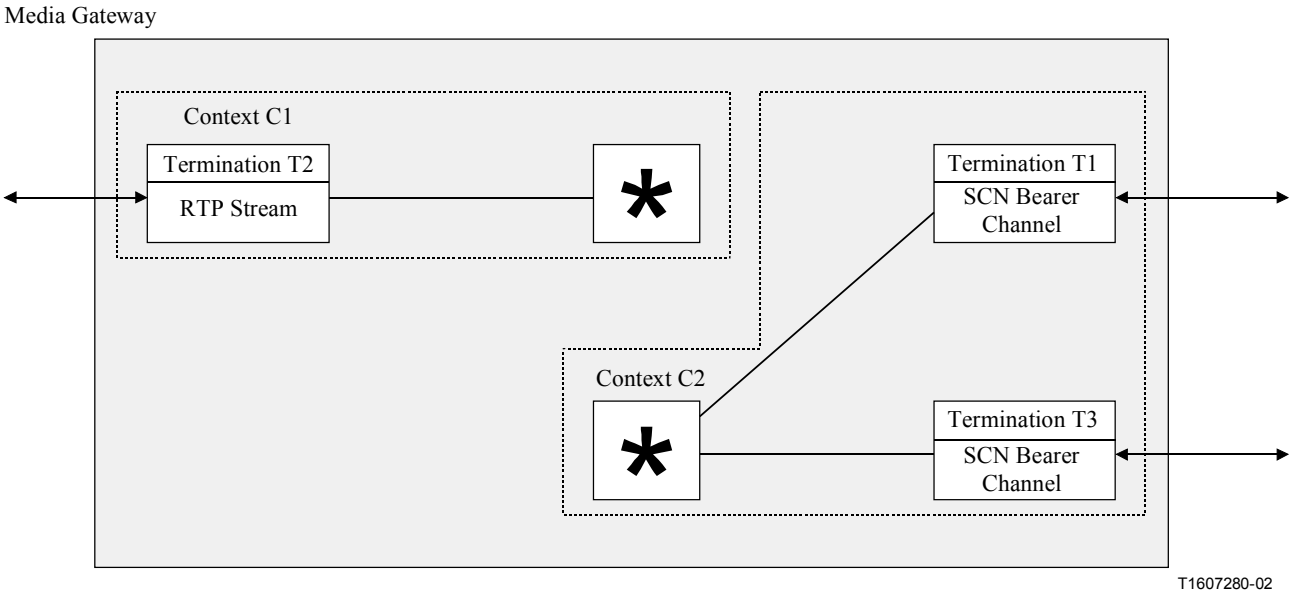

Figure 3/H.248.1 – Example Call Waiting Scenario/Answer by T1

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 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 (who hears/sees whom)

  The topology of a Context describes the flow of media between the Terminations within a Context. In contrast, the mode of a Termination (send/receive/…) describes the flow of the media at the ingress/egress 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.

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 is taken out of the call it is in.

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 MultiplexDescriptor 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 StreamDescriptors 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.

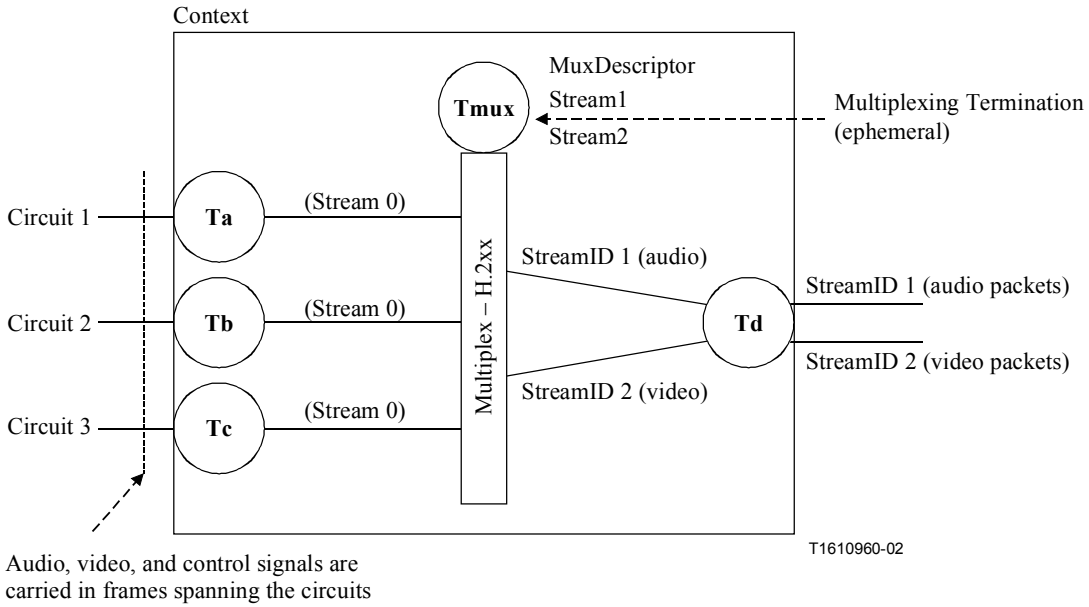

Figure 4/H.248.1 – Multiplexed Termination Scenario – Circuit to Packet

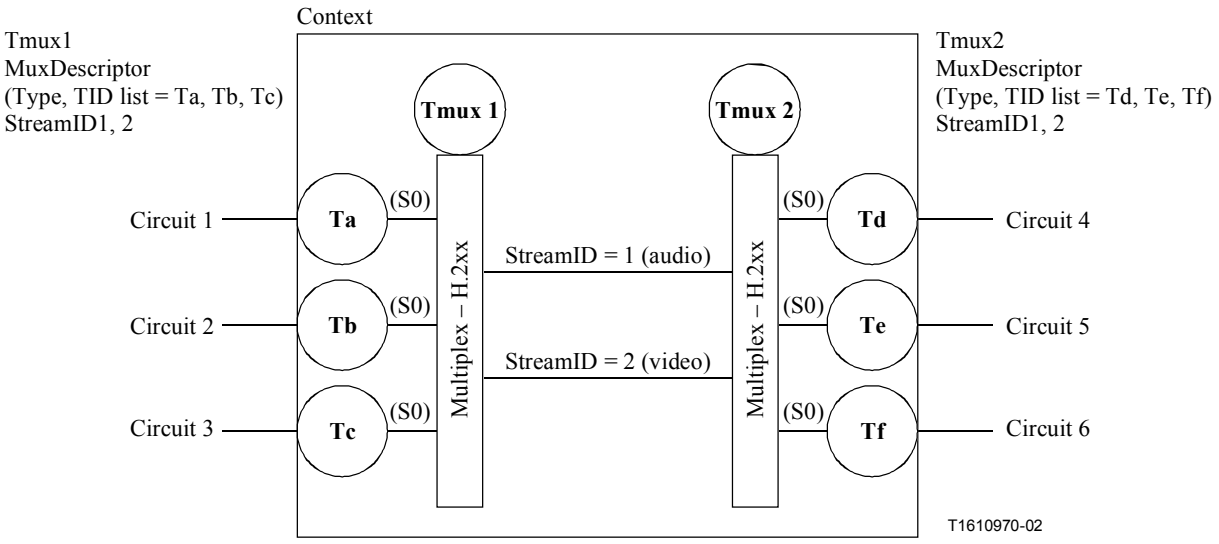

Figure 5/H.248.1 – Multiplexed Termination Scenario – Circuit to Circuit

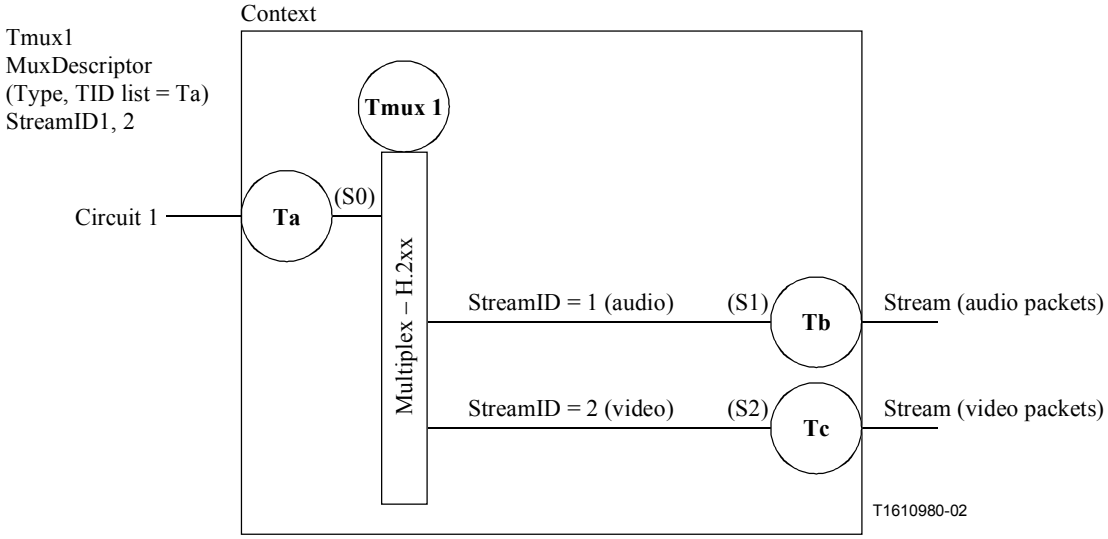

Figure 6/H.248.1 – Multiplexed Termination Scenario – Single to Multiple Terminations

Terminations may be created which represent multiplexed bearers, such as an ATM AAL Type 2 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 it has 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.

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. 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 a 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 response may be particularly useful in the Audit commands.

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

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, StatisticsIds 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 support all properties, signals, events and statistics defined in a package. It must also support all Signal and Event parameters. The MG may support a subset of the values listed in a package for a particular Property or Parameter.

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 Package 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 "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. 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 Termination state 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 PackageName 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.

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 behavior. For more details, see 7.1 dealing with the individual Descriptors.

The following table lists all 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
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

Descriptor name	Description
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 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 service change occurred, etc.
ObservedEvents	In Notify or AuditValue, report of events observed
Statistics	In Subtract and Audit, report of Statistics kept on a Termination
Topology	Specifies flow directions between Terminations in a Context
Error	Contains an error code and optionally error text; it may occur in command replies and in Notify requests

6.2.5 Root Termination

Occasionally, a command must refer to the entire gateway, 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 and statistics (signals are not appropriate for root). Accordingly, the root TerminationID may appear in:

- a Modify command – to change a property or set an event;
- a Notify command – to report an event;
- an AuditValue return – to examine the values of properties and statistics implemented on root;
- an AuditCapability – to determine what properties of root are implemented;
- a ServiceChange – to declare the 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.

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.

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;
- 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}
```

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 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 three subsidiary descriptors: LocalControl, Local, and Remote. The relationship between these descriptors is thus:

Media descriptor

TerminationState Descriptor

Stream descriptor

LocalControl descriptor

Local descriptor

Remote descriptor

As a convenience, LocalControl, Local, or Remote descriptors may be included in the Media descriptor without an enclosing Stream descriptor. In this case, the StreamID is assumed to be 1.

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", "out of service", or "in service". The "test" state indicates that the Termination is being tested. The state "out of service" indicates that the Termination cannot be used for traffic. The state "in service" indicates that a Termination can be used or is being used for normal traffic. "in service" 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, any 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.

7.1.6 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 underspecified as in 7.1.1.

The allowed values for the mode property are send-only, receive-only, send/receive, inactive and loop-back. "Send" and "receive" 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. The default value for the mode property is "Inactive". Signals and Events are not affected by mode.

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 not 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/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, is SHALL respond with error 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.

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).

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 (IETF RFC 2327). In session descriptions sent from the MGC to the MG, the following exceptions to the syntax of IETF 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 IETF RFC 2327. 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 any of the requested property group alternatives that it can currently support. If ReserveValue is True, the MG reserves the resources required to support any 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 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 EventsDescriptor parameter contains a RequestIdentifier and a list of events that the Media Gateway is requested to detect and report. The RequestIdentifier 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 RequestIdentifier is omitted if the EventsDescriptor is empty (i.e. no events are specified).

Each event in the descriptor contains the Event name, an optional streamID, an optional KeepActive flag, and optional parameters. The Event name consists of a Package Name (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 digit map completion event is present or implied in the EventsDescriptor, the EventDM parameter is used to carry either the name or the value of the associated digit map. 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 digit map (see 7.1.14). Any other action is for further study. 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 EventsDescriptor to be loaded. A new EventsDescriptor can be loaded either as the result of receiving a command with a new EventsDescriptor, or by activating an embedded EventsDescriptor.

If EventBufferControl equals Off, the MG continues processing based on the active EventsDescriptor.

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

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

When the MG receives a command with a new EventsDescriptor, 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 EventsDescriptor, 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 EventsDescriptor.

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 EventsDescriptor, 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 EventsDescriptor. While waiting for a new EventsDescriptor, any events detected that appear in the EventsBufferDescriptor will be placed in the EventBuffer. When a new EventsDescriptor is received, the event processing will repeat from step 1).

   b) If the event is not in the new EventsDescriptor, the MG SHALL discard the event and repeat from step 1).

Case 2

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

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 Recommendation of the H.248.x series for specific transport considerations.

The default value of EventBufferControl is Off.

NOTE 2 – Since the EventBufferControl property is in the TerminationStateDescriptor, the MG might receive a command that changes the EventBufferControl property and does not include an EventsDescriptor.

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.

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 EventsDescriptors with an event both marked KeepActive and containing an embedded SignalsDescriptor.

Only one level of embedding is permitted. An embedded EventsDescriptor SHALL NOT contain another embedded EventsDescriptor; an embedded EventsDescriptor MAY contain an embedded SignalsDescriptor.

An EventsDescriptor 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 EventsDescriptor executes, shall be processed according to the new EventsDescriptor.

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 events buffer 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 Continuity Test package. Signals may also request preparation of media content for future signals.

A SignalsDescriptor is a parameter that contains the set of signals that the Media Gateway is asked to apply to a Termination. A SignalsDescriptor contains a number of signals and/or sequential signal lists. A SignalsDescriptor 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 Package name (in which the signal is defined) and a SignalID. No wildcard shall be used in the SignalID. Signals that occur in a SignalsDescriptor 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.

Finally, the optional parameter "notifyCompletion" 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 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 duration is an integer value that is expressed in hundredths of a second.

There are three types of signals:

- on/off – the signal lasts until it is turned off;
- timeout – the signal lasts until it is turned off or a specific period of time elapses;
- brief – 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 SignalsDescriptor, 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 signal list identifier 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.

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

Signals are defined as proceeding from the Termination towards the exterior of the Context unless otherwise specified in a package. 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 SignalsDescriptor, or detection of an Event on the Termination (see 7.1.9).

A new SignalsDescriptor replaces any existing SignalsDescriptor. 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 flagshall be continued if they are currently playing and have not already completed. If a replacement signal 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 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
Mux
Events
Media
Signals
ObservedEvents
DigitMap
Statistics
Packages
EventBuffer

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

7.1.13 ServiceChange descriptor

The ServiceChangeDescriptor contains the following parameters:

- ServiceChangeMethod
- ServiceChangeReason
- ServiceChangeAddress
- ServiceChangeDelay
- ServiceChangeProfile
- ServiceChangeVersion
- ServiceChangeMGCId
- TimeStamp
- Extension

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 digit map may be preloaded into the MG by management action and referenced by name in an EventsDescriptor, may be defined dynamically and subsequently referenced by name, or the actual digitmap itself may be specified in the EventsDescriptor. It is permissible for a digit map 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 DigitMapDescriptor 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 EventsDescriptors specifying the name, including any EventsDescriptor 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 having been dialed. 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 digit map, 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 digit map, 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 digit map 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 digit map symbols or as a regular expression of digit map symbols. These digit map 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 digit map is being applied. (The mapping between events and digit map 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 digit map. 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. 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.

7.1.14.4 DigitMap Completion Event

A digit map is active while the Events descriptor which invoked it is active and it has not completed. A digit map completes when:

- a timer has expired; or
- an alternative event sequence has been matched and no other alternative event sequence in the digit map 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 digit map will be impossible no matter what additional events are received.

Upon completion, a digit map completion event as defined in the package providing the events being mapped into the digit map shall be generated. At that point the digit map is deactivated. Subsequent events in the package are processed as per the currently active event processing mechanisms.

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 digit map.

2) At each step, 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.

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 per the currently active event processing mechanisms.

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

7.1.14.6 DigitMap Activation

A digit map is activated whenever a new Event descriptor is applied to the Termination or embedded Event descriptor is activated, that Event descriptor contains a digit map completion event. The digit map completion event contains an eventDM field in the requested actions field. Each new activation of a digit map 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 digit map 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 EventsDescriptor, a MG shall respond with an error reponse, including Error 457 (Missing parameter in signal or event).

7.1.14.7 Interaction Of DigitMap and Event Processing

While the digit map is activated, detection is enabled for all events defined in the package containing the specified digit map 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 digit map 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 digit map completion event.

7.1.14.8 Wildcards

Note that if a package contains a digit map completion event, then an event specification consisting of the package name with a wildcarded ItemID (Property Name) will activate a digit map; 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 digit map:

```
(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 within a specific Context. There is a set of standard statistics kept for each Termination where appropriate (number of octets sent and received for example). The particular statistical properties that are reported for a given Termination are determined by the Packages realized by the Termination. By default, statistics are reported when the Termination is Subtracted from the Context. This behaviour can be overridden by including an empty AuditDescriptor in the Subtract command. 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. Statistics are reset when a Termination is Subtracted from a Context.

7.1.16 Packages descriptor

Used only with the AuditValue command, the PackageDescriptor 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 ObservedEventsDescriptor returns events in the event buffer which have not been Notified. ObservedEvents contains the RequestIdentifier of the EventsDescriptor 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 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 triples of the form (*T1*, *T2*, *association*). *T1* and *T2* specify Terminations within the Context, possibly using the ALL or CHOOSE wildcard. 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 both *T1* and *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.

Figure 7 and the table 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.

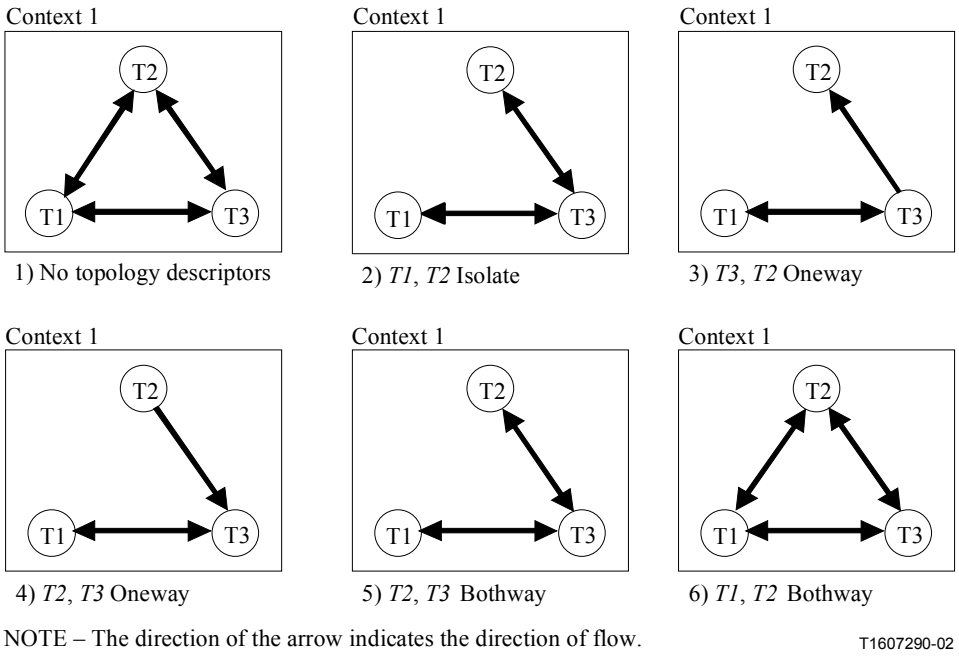

Figure 7/H.248.1 – Example topologies

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 A 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 A 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 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.

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.

7.1.19 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-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 syntactical 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 clauses. 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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

```
Add( TerminationID
     [, MediaDescriptor]
     [, ModemDescriptor]
     [, MuxDescriptor]
     [, EventsDescriptor]
     [, EventBufferDescriptor]
     [, SignalsDescriptor]
     [, DigitMapDescriptor]
     [, AuditDescriptor]
)
```

The TerminationID specifies the Termination to be added to the Context. The Termination is either created, or taken from the null Context. If a CHOOSE wildcard is used in the TerminationID, the selected TerminationID will be returned. 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 MediaDescriptor describes all media streams.

The optional ModemDescriptor and MuxDescriptor specify a modem and 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 EventsDescriptor parameter is optional. If present, it provides the list of events that should be detected on the Termination.

The EventBufferDescriptor 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 SignalsDescriptor parameter is optional. If present, it provides the list of signals that should be applied to the Termination.

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

The AuditDescriptor is optional. If present, the command will return descriptors as specified in the AuditDescriptor.

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

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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

Modify(TerminationID

[, MediaDescriptor]

[, ModemDescriptor]

[, MuxDescriptor]

[, EventsDescriptor]

[, EventBufferDescriptor]

[, SignalsDescriptor]

[, DigitMapDescriptor]

[, AuditDescriptor]

)

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 option 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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

Subtract(TerminationID

[, AuditDescriptor]

)

TerminationID in the input parameters represents the Termination 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, the Statistics parameter 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 AuditDescriptor is optional. If present, the command will return only those descriptors as specified in the AuditDescriptor, which may be empty. If 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 property values shall revert to:

- the default value, if specified for the property 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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

Move(TerminationID

[, MediaDescriptor]

[, ModemDescriptor]

[, MuxDescriptor]

[, EventsDescriptor]

[, EventBufferDescriptor]

[, SignalsDescriptor]

[, DigitMapDescriptor]

[, AuditDescriptor]

)

The TerminationID specifies the Termination to be moved. It may be wildcarded, but CHOOSE shall not be used in the TerminationID. 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 AuditDescriptor with the Statistics option, 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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,DigitMapDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

[,PackagesDescriptor]

AuditValue(TerminationID,

AuditDescriptor

)

TerminationID 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. This convention may reduce the volume of data required to audit a group of Terminations. Use of CHOOSE is an error.

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 ObservedEventsDescriptor is audited while a DigitMap is active, the returned ObservedEvents descriptor also includes a digit map 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 EventBufferDescriptor. PackagesDescriptor returns a list of packages realized by the Termination. DigitMapDescriptor 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 on the Termination. 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 wildcard. 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=*{W-AuditValue=t1/*{Packages{aaa,bbb}}}

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

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 and the Context to which they are associated
All	Root	List of all ContextIds (the ContextID list should be returned by using multiple action replies, each containing a ContextID from the list)
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.

TerminationID

[,MediaDescriptor]

[,ModemDescriptor]

[,MuxDescriptor]

[,EventsDescriptor]

[,SignalsDescriptor]

[,ObservedEventsDescriptor]

[,EventBufferDescriptor]

[,StatisticsDescriptor]

AuditCapabilities(TerminationID,

AuditDescriptor)

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 matching the wildcard. This convention may reduce the volume of data required to audit a group of Terminations.

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

The EventsDescriptor returns the list of possible events on the Termination together with the list of all possible values for the EventsDescriptor Parameters. EventBufferDescriptor returns the same information as EventsDescriptor. The SignalsDescriptor 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. StatisticsDescriptor returns the names of the statistics being kept on the termination. ObservedEventsDescriptor 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 AuditCapabilties 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 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 parameter specifies the Termination issuing the Notify Command. The TerminationID shall be a fully qualified name.

The ObservedEventsDescriptor 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 0 are for further study.

Notify Commands with RequestID not equal to 0 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 EventsDescriptor 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 EventsDescriptor or embedded events descriptor unless the RequestID is 0 (which is for further study).

The ErrorDescriptor 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.

TerminationID,

[ServiceChangeDescriptor]

ServiceChange(TerminationID,

ServiceChangeDescriptor

)

The TerminationID parameter 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.

The ServiceChangeDescriptor contains the following parameters as required:

- ServiceChangeMethod;
- ServiceChangeReason;
- ServiceChangeDelay;
- ServiceChangeAddress;
- ServiceChangeProfile;
- ServiceChangeVersion;
- ServiceChangeMgcId;
- TimeStamp.

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 Termination 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 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 preprovisioned list). Since MG state may have changed, the MGC may wish to use the Audit command to resynchronize its state with the MG's.
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.

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.

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 ServiceChangeDescriptor or the ServiceChangeResultDescriptor. 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.

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 "out of service" until the Termination is in the null Context.

The optional ServiceChangeProfile parameter specifies the Profile (if any) of the protocol supported. The ServiceChangeProfile includes the version of the profile supported.

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

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.

The optional Extension parameter may contain any value whose meaning is mutually understood by the MG and MGC.

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 also announce a registration command by specifying the "Root" for the TerminationID and ServiceChangeMethod equal to Failover when the MG detects MGC failures. 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 in the following paragraph. The MG may specify the transport ServiceChangeAddress to be used by the MGC for sending messages in the ServiceChangeAddress parameter in the input ServiceChangeDescriptor. 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.

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;
- ServiceChangeVersion, if the responder wishes to negotiate the version of the protocol to be used for the association.

The following ServiceChangeReasons are defined. This list may be extended by an IANA registration as outlined in 13.3.

```
900 Service Restored
901 Cold Boot
902 Warm Boot
903 MGC Directed Change
904 Termination malfunctioning
905 Termination taken out of service
906 Loss of lower layer connectivity (e.g. downstream sync)
907 Transmission Failure
908 MG Impending Failure
909 MGC Impending Failure
910 Media Capability Failure
911 Modem Capability Failure
912 Mux Capability Failure
913 Signal Capability Failure
914 Event Capability Failure
915 State Loss
```

7.2.9 Manipulating and Auditing Context Attributes

The commands of the protocol as discussed in the preceding clauses apply to Terminations. This clause specifies how Contexts are manipulated and audited.

Commands are grouped into actions (see clause 8). An action applies to one Context. In addition to commands, an action may contain Context manipulation and auditing instructions.

An action request sent to a MG may include a request to audit attributes of a Context. 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 which 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 IETF 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.

7.3 Command Error Codes

Errors consist of an IANA registered error code and an explanatory string. Sending the explanatory string is optional. Implementations are encouraged to append diagnostic information to the end of the string.

When a MG reports an error to a MGC, it does so in an error descriptor. An error descriptor consists of an error code and optionally the associated explanatory string.

ITU-T Rec. H.248.8 contains the error codes supported by Recommendations in the H.248.x series.

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 8 is a graphic representation of the Transaction, Action and Command relationships.

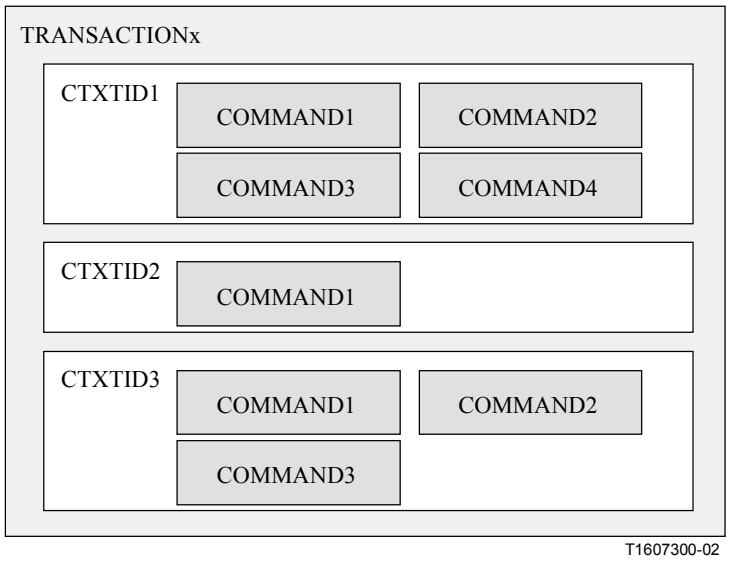

Figure 8/H.248.1 – Transactions, Actions and Commands

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.

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. 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 TransactionID 0 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 clauses.

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 and one or more Commands per 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 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. 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.

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

A SignalsDescriptor 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 EventsDescriptor or EventBufferDescriptor 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 Transaction request. 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 7.1.19 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, the requested Action response will consist of the Context ID 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, it will process them 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 Transaction Request).

If the end of a transaction cannot be reliably determined and one or more Actions can be parsed, it will process them and then return 403 (Syntax Error in Transaction Request) as the last action reply for the transaction. If no Actions can be parsed, it will return 403 (Syntax Error in Transaction Request) 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 root (normalMGExecutionTime) is settable by the MGC to indicate the interval within which the MGC expects a response to any transaction from the MG. 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. 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.

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 for the present version of the protocol.

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 Recommendations of the H.248.x series. Additional Transports may be defined as additional Recommendations of the H.248.x series. 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 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 EventsDescriptor.
- 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 all were to 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 0 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, in average, 10% of the lines will be busy, placing calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 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 5 to 6 transactions for each Termination, every 30 minutes on average, or, to put it otherwise, about one transaction per Termination every 5 to 6 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.

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 (IETF RFC 2401-2411).

The AH header (IETF 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 (IETF 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 clause 5 of IETF RFC 2406, which defines a minimum set of algorithms for integrity checking and encryption. Similarly, implementations employing the AH header SHALL comply with clause 5 of IETF RFC 2402, which defines a minimum set of algorithms for integrity checking using manual keys.

Implementations SHOULD use IKE (IETF 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 IETF RFC 2402. The semantics of the header fields are the same as the "transport mode" of IETF 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 clause 5 of IETF 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: either 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. 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 MG (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. 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 the protocol. 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 EventsDescriptors 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 preprovisioned by a management mechanism, outside the scope of this protocol, 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 Transaction Reply not including a ServiceChangeMgcId parameter. If the MGC does not accept the MG's registration, it sends a Transaction Reply, providing the address of an alternate MGC to be contacted by including a ServiceChangeMgcId parameter.

If the MG receives a Transaction Reply 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 preprovisioned 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 505 (Transaction Request Received before a ServiceChange Reply has been received).

11.3 Negotiation of protocol version

The first ServiceChange command from a MG shall contain the version number of the protocol supported by the MG in the ServiceChangeVersion parameter. Upon receiving such a message, if the MGC supports only a lower version, then the MGC shall send a ServiceChangeReply 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 406 (Version Not Supported).

If the MGC supports a higher version than the MG but is able to support the lower version proposed by the MG, it shall send a ServiceChangeReply with the lower version and thereafter all the messages between MG and MGC shall conform to the lower version of the protocol. If the MGC is unable to comply, it shall reject the association, with Error 406 (Version Not Supported).

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" method.

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" method.

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 preprovisioned 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 the 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 state on 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, 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.

12 Package definition

The primary mechanism for extension is by means of Packages. Packages define additional Properties, 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 Recommendations of the H.248.x series).

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 13.2. 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: togen/playtone is the text encoding to refer to the play tone signal in the tone generation package.

A Package will contain the following sections:

12.1.1 Package

Overall description of the package, specifying:

Package Name: only descriptive

PackageID: is an identifier

Description:

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 SHOULD NOT publish this PackageID when reporting packages.

Extends (Optional): existing package Descriptor

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.

12.1.2 Properties

Properties defined by the package, specifying:

Property Name: only descriptive

PropertyID: is an identifier

Description:

Type: One of:

Boolean

String: UTF-8 string

Octet String: A number of octets. See Annexes 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 (see 12.3)

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 clause (with the exception of sub-list). For example, Type: sublist of enumeration. The encoding of sub-lists is specified in Annexes A and B.3.

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

Defined in:

Which H.248.1 descriptor the property is defined in. LocalControl is for stream dependent properties. TerminationState is for stream independent properties. These are

expected to be the most common cases, but it is possible for properties to be defined in other descriptors.

Characteristics: Read/Write or both, 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.

12.1.3 Events

Events defined by the package, specifying:

Event name: only descriptive

EventID: is an identifier

Description:

EventsDescriptor Parameters:

Parameters used by the MGC to configure the event, and found in the EventsDescriptor. See 12.2.

ObservedEventsDescriptor Parameters:

Parameters returned to the MGC in Notify requests and in replies to command requests from the MGC that audit ObservedEventsDescriptor, and found in the ObservedEventsDescriptor. See 12.2.

12.1.4 Signals

Signals defined by the package, specifying:

Signal Name: only descriptive

SignalID: is an identifier. SignalID is used in a SignalsDescriptor

Description

SignalType: one of:

OO (On/Off)

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 StatisticsDescriptor

Description:

Units: unit of measure, e.g. milliseconds, packets

12.1.6 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" or "DigitMap".

Type: One of:

Boolean

String: UTF-8 octet string

Octet String: A number of octets. See Annexes 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 (see 12.3)

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 clause (with the exception of sub-list). For example, Type: sub-list of enumeration. The encoding of sub-lists is specified in Annexes A and B.3.

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 behavior when the value is omitted from its descriptor. For example, a package may specify that procedures related to the parameter are suspended when it value is omitted. A default value (but not procedures) may be specified as provisionable.

Description:

12.3 Lists

Possible values for parameters include enumerations. Enumerations may be defined in a list. It is recommended that the list be IANA registered so that packages that extend the list can be defined without concern for conflicting names.

12.4 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.

12.5 Package registration

A package can be registered with IANA for interoperability reasons. See clause 13 for IANA considerations.

13 IANA considerations

13.1 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

13.2 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 is 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.

13.3 ServiceChange reasons

The following considerations SHALL be met to register service change 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 is detected 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.

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*.

A.1 Coding of wildcards

The use of wildcards ALL and CHOOSE is allowed in the protocol. This allows a MGC to partially specify Termination IDs and to let the MG choose from the values that conform to the partial specification. Termination IDs 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 Termination IDs. This means that Termination IDs consist of at most 64 bits. A fully specified Termination ID 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 Termination ID 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 Termination IDs are three octets long and that each octet represents a level in a hierarchical naming scheme. A valid Termination ID is:

00000001 00011110 01010101.

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

wildcarding field: 10000111

Termination ID: 00000001 00011110 xxxxxxxx.

The values of the bits labeled "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

Termination ID: 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

Termination ID: 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.

Context IDs may also be wildcarded. In the case of Context IDs, however, specifying partial names is not allowed. Context ID 0x0 SHALL be used to indicate the NULL Context, Context ID 0xFFFFFFFE SHALL be used to indicate a CHOOSE wildcard, and Context ID 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 series defining the transport mechanism for the definition that applies in that case.

NOTE 2 – The ASN.1 specification below contains a clause defining `TerminationIDList` as a sequence of TerminationIDs. The length of this sequence SHALL be one, except possibly when used in contextAuditResult.

NOTE 3 – 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 4 – 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 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 1.
     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
     ...
}

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
     },
     ...
}

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

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

ErrorCode ::= INTEGER(0..65535)
-- See clause 13 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,
     ...
}

ContextAttrAuditRequest ::= SEQUENCE
{
     topology  NULL OPTIONAL,
     emergency NULL OPTIONAL,
     priority  NULL OPTIONAL,
     ...
}

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)
     },
     ...
}

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,
...
}


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

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

AuditRequest ::= SEQUENCE
{
     terminationID           TerminationID,
     auditDescriptor         AuditDescriptor,
     ...
}

AuditReply ::= CHOICE
{
     contextAuditResult      TerminationIDList,
     error                   ErrorDescriptor,
     auditResult             AuditResult,
     ...
}
AuditResult ::= SEQUENCE
{

     terminationID           TerminationID,
     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,
     ...
}
```

```
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 extraInfo 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.
-- Termination ID 0xFFFFFFFFFFFFFFFF indicates the ROOT Termination.

TerminationIDList ::= SEQUENCE OF TerminationID

MediaDescriptor ::= SEQUENCE
{

     termStateDescrTerminationStateDescriptor 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,
     ...
}

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 Statistics ID. (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 Property, Event, Signal, or Statistics ID, use
-- 0xFFFF for last two octets, choose is not allowed.
-- Wildcarding of Package Name is permitted only if Property,
-- Event, Signal, or Statistics ID 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),
     ...
}

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,
     ...
}
```

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

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,
     ...
}

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

SignalName ::= PkgdName

NotifyCompletion ::= BIT STRING
{
     onTimeOut(0), onInterruptByEvent(1),
     onInterruptByNewSignalDescr(2), otherReason(3)
}
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 digit map syntax
     ...
}

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,
...
}
```

```
ServiceChangeAddress ::= CHOICE
{
     portNumber              INTEGER(0..65535),     -- TCP/UDP port number
     ip4Address              IP4Address,
     ip6Address              IP6Address,
     domainName              DomainName,
     deviceName              PathName,
     mtpAddress              OCTET STRING(SIZE(2)),
     ...
}

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 verison 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
}

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 8601:1988
}

Value ::= SEQUENCE OF OCTET STRING


END
```

A.3 Digit maps and path names

From a syntactic viewpoint, digit maps are strings with syntactic restrictions imposed upon them. The syntax of valid digit maps is specified in ABNF (IETF RFC 2234). The syntax for digit maps 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              ;digits 0-9
              / %x41-4B / %x61-6B    ;a-k and A-K
              / "L"     / "S"              ;Inter-event timers
                                              ;(long, short)
              / "Z"                          ;Long duration event
DOT = %x2E ; "."
LWSP = *(WSP / COMMENT / EOL)
WSP = SP / HTAB
COMMENT = ";" *(SafeChar / RestChar / WSP) EOL
EOL = (CR [LF]) / LF
SP = %x20
HTAB = %x09
CR = %x0D
LF = %x0A
SafeChar = DIGIT / ALPHA / "+" / "-" / "&" / "!" / "_" / "/" /
       "'" / "?" / "@" / "^" / "`" / "~" / "*" / "$" / "\" /
       "(" / ")" / "%" / "."
RestChar = ";" / "[" / "]" / "{" / "}" / ":" / "," / "#" /
       "<" / ">" / "=" / %x22
```

```
DIGIT = %x30-39          ; digits 0 through 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 IETF 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, digitmap Ids etc. SDP is case sensitive as per IETF RFC 2327.

```
; NOTE – The ABNF in this clause 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 1.

messageBody          = ( errorDescriptor / transactionList )

transactionList      = 1*( transactionRequest / transactionReply /
                       transactionPending / transactionResponseAck )
;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
contextProperty     = (topologyDescriptor / priority / EmergencyToken)

contextAudit        = ContextAuditToken LBRKT
                      contextAuditProperties *(COMMA
                      contextAuditProperties) RBRKT

; at-most-once
contextAuditProperties = ( TopologyToken / EmergencyToken /
                           PriorityToken )

; "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 LBRKT
                      [ ImmAckRequiredToken COMMA]
               ( errorDescriptor / actionReplyList ) RBRKT

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
                      TerminationID [LBRKT ammParameter *(COMMA
                      ammParameter) RBRKT]

;at-most-once
ammParameter        = (mediaDescriptor / modemDescriptor /
                        muxDescriptor / eventsDescriptor /
                        signalsDescriptor / digitMapDescriptor /
                        eventBufferDescriptor / auditDescriptor)

ammsReply           = (AddToken / MoveToken / ModifyToken /
                        SubtractToken ) EQUAL TerminationID [ LBRKT
                        terminationAudit RBRKT ]

subtractRequest     = SubtractToken EQUAL TerminationID
                        [ LBRKT auditDescriptor RBRKT]

auditRequest        = (AuditValueToken / AuditCapToken ) EQUAL
                      TerminationID LBRKT auditDescriptor RBRKT
```

```
auditReply          = (AuditValueToken / AuditCapToken )
                      ( contextTerminationAudit  / auditOther)

auditOther          = EQUAL TerminationID [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 / auditItem)

auditDescriptor     = AuditToken LBRKT [ auditItem
                      *(COMMA auditItem) ] RBRKT

notifyRequest       = NotifyToken EQUAL TerminationID
                      LBRKT ( observedEventsDescriptor
                            [ COMMA errorDescriptor ] ) RBRKT

notifyReply         = NotifyToken EQUAL TerminationID
                      [ LBRKT errorDescriptor RBRKT ]

serviceChangeRequest = ServiceChangeToken EQUAL TerminationID
                       LBRKT serviceChangeDescriptor RBRKT

serviceChangeReply  = ServiceChangeToken EQUAL TerminationID
                      [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.
ContextID           = (UINT32 / "*" / "-" / "$")

domainAddress       = "[" (IPv4address / IPv6address) "]"
;RFC2373 contains the definition of IP6Addresses.
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

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 / "-" / "*" / ".")

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 )

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
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
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 ( "OFF" / LockStepToken )

terminationStateDescriptor = TerminationStateToken LBRKT
           terminationStateParm *( COMMA terminationStateParm ) RBRKT

; at-most-once per item except for propertyParm
terminationStateParm =(propertyParm / serviceStates / eventBufferControl )

serviceStates       = ServiceStatesToken EQUAL ( TestToken /
                      OutOfSvcToken / InSvcToken )

muxDescriptor       = MuxToken EQUAL MuxType  terminationIDList

MuxType             = ( H221Token / H223Token / H226Token / V76Token
                       / extensionParameter )

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 ]

; at-most-once each of KeepActiveToken , eventDM 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 )

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, eventDM or
; eventStream
; KeepActiveToken and embedSig must not both be present
secondEventParameter = ( embedSig / KeepActiveToken / eventDM /
                         eventStream / eventOther )

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, every signalParameterName at most once
sigParameter        = sigStream / sigSignalType / sigDuration / sigOther
                    / notifyCompletion / KeepActiveToken
sigStream           = StreamToken EQUAL StreamID
sigOther            = sigParameterName parmValue
sigParameterName    = NAME
sigSignalType       = SignalTypeToken EQUAL signalType
signalType          = (OnOffToken / TimeOutToken / BriefToken)
sigDuration         = DurationToken EQUAL UINT16
notifyCompletion    = NotifyCompletionToken EQUAL (LBRKT
               notificationReason *(COMMA notificationReason) RBRKT)

notificationReason   = ( TimeOutToken / InterruptByEventToken
                         / InterruptByNewSignalsDescrToken
                         / OtherReasonToken )
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] 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"   ;Inter-event timers (long, short)
                    / "Z"         ;Long duration modifier

;at-most-once, and DigitMapToken and PackagesToken are not allowed
;in AuditCapabilities command
auditItem           = ( MuxToken / ModemToken / MediaToken /
                        SignalsToken / EventBufferToken /
                        DigitMapToken / StatsToken / EventsToken /
                        ObservedEventsToken / PackagesToken )

serviceChangeDescriptor = ServicesToken LBRKT serviceChangeParm
                          *(COMMA serviceChangeParm) RBRKT

; each parameter at-most-once
; 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 )

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 8601:1988
; 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]

topologyDescriptor  = TopologyToken LBRKT topologyTriple
                      *(COMMA topologyTriple) RBRKT
topologyTriple      = terminationA COMMA
                      terminationB COMMA topologyDirection
terminationA        = TerminationID
terminationB        = TerminationID
topologyDirection   = BothwayToken / IsolateToken / OnewayToken

priority            = PriorityToken EQUAL UINT16

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 / RestChar/ WSP) DQUOTE
```

```
UINT16              = 1*5(DIGIT)  ; %x0-FFFF
UINT32              = 1*10(DIGIT) ; %x0-FFFFFFFF

NAME                = ALPHA *63(ALPHA / DIGIT / "_" )
VALUE               = quotedString / 1*(SafeChar)
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")
AuditToken              = ("Audit"                  / "AT")
AuditCapToken           = ("AuditCapability"        / "AC")
AuditValueToken         = ("AuditValue"             / "AV")
AuthToken               = ("Authentication"         / "AU")
BothwayToken            = ("Bothway"                / "BW")
BriefToken              = ("Brief"                  / "BR")
BufferToken             = ("Buffer"                 / "BF")
CtxToken                = ("Context"                / "C")
ContextAuditToken       = ("ContextAudit"           / "CA")
DigitMapToken           = ("DigitMap"               / "DM")
DisconnectedToken       = ("Disconnected"           / "DC")
DelayToken              = ("Delay"                  / "DL")
DurationToken           = ("Duration"               / "DR")
EmbedToken              = ("Embed"                  / "EM")
EmergencyToken          = ("Emergency"              / "EG")
ErrorToken              = ("Error"                  / "ER")
EventBufferToken        = ("EventBuffer"            / "EB")
EventsToken             = ("Events"                 / "E")
FailoverToken           = ("Failover"               / "FL")
ForcedToken             = ("Forced"                 / "FO")
GracefulToken           = ("Graceful"               / "GR")
H221Token               = ("H221" )
H223Token               = ("H223" )
H226Token               = ("H226" )
HandOffToken            = ("HandOff"                / "HO")
```

```
ImmAckRequiredToken       = ("ImmAckRequired"        / "IA")
InactiveToken             = ("Inactive"              / "IN")
IsolateToken              = ("Isolate"               / "IS")
InSvcToken                = ("InService"             / "IV")
InterruptByEventToken     = ("IntByEvent"            / "IBE")
InterruptByNewSignalsDescrToken
                          = ("IntBySigDescr"         / "IBS")
KeepActiveToken           = ("KeepActive"            / "KA")
LocalToken                = ("Local"                 / "L")
LocalControlToken         = ("LocalControl"          / "O")
LockStepToken             = ("LockStep"              / "SP")
LoopbackToken             = ("Loopback"              / "LB")
MediaToken                = ("Media"                 / "M")
MegacopToken              = ("MEGACO"                / "!")
MethodToken               = ("Method"                / "MT")
MgcIdToken                = ("MgcIdToTry"            / "MG")
ModeToken                 = ("Mode"                  / "MO")
ModifyToken               = ("Modify"                / "MF")
ModemToken                = ("Modem"                 / "MD")
MoveToken                 = ("Move"                  / "MV")
MTPToken                  = ("MTP")
MuxToken                  = ("Mux"                   / "MX")
NotifyToken               = ("Notify"                / "N")
NotifyCompletionToken     = ("NotifyCompletion"      / "NC")
ObservedEventsToken       = ("ObservedEvents"        / "OE")
OnewayToken               = ("Oneway"                / "OW")
OnOffToken                = ("OnOff"                 / "OO")
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")
RestartToken              = ("Restart"               / "RS")
RemoteToken               = ("Remote"                / "R")
ReservedGroupToken        = ("ReservedGroup"         / "RG")
ReservedValueToken        = ("ReservedValue"         / "RV")
SendonlyToken             = ("SendOnly"              / "SO")
SendrecvToken             = ("SendReceive"           / "SR")
ServicesToken             = ("Services"              / "SV")
ServiceStatesToken        = ("ServiceStates"         / "SI")
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 of 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".

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 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 1.

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 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
Encryptkey	100A	Octet string size (0..65535)	Encryption key Ref.: ITU-T Rec. H.235
Echocanc	100B		Not Used. See 1 E.13 for an example of possible Echo Control properties.
Gain	100C	Unsigned integer	Gain in dB: 0..65535
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.: IETF RFC 1890

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

PropertyID	Property tag	Type	Value
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
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

<table>
<tr><th>PropertyID</th><th>Property tag</th><th>Type</th><th colspan="2">Value</th></tr>
<tr><td>APPCDV</td><td>4014</td><td>24-bit integer</td><td colspan="2">Acceptable P-P CDV
Ref.: ATM Forum UNI 4.0</td></tr>
<tr><td>CPPCDV</td><td>4015</td><td>24-bit integer</td><td colspan="2">Cumulative P-P CDV
Ref.: ATM Forum UNI 4.0</td></tr>
<tr><td>ACLR</td><td>4016</td><td>8-bit integer</td><td colspan="2">Acceptable Cell Loss Ratio
Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0</td></tr>
<tr><td>MEETD</td><td>4017</td><td>16-bit integer</td><td colspan="2">Maximum End-to-end transit delay
Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0</td></tr>
<tr><td>CEETD</td><td>4018</td><td>16-bit integer</td><td colspan="2">Cumulative End-to-end transit delay
Ref.: ITU-T Rec. Q.2965.2, ATM Forum UNI 4.0</td></tr>
<tr><td rowspan="9">QosClass</td><td rowspan="9">4019</td><td rowspan="9">Integer 0-5</td><td colspan="2">QoS Class</td></tr>
<tr><td>QoS Class</td><td>Meaning</td></tr>
<tr><td>0</td><td>Default QoS associated with the ATC as defined in ITU-T Rec. Q.2961.2</td></tr>
<tr><td>1</td><td>Stringent</td></tr>
<tr><td>2</td><td>Tolerant</td></tr>
<tr><td>3</td><td>Bi-level</td></tr>
<tr><td>4</td><td>Unbounded</td></tr>
<tr><td>5</td><td>Stringent Bi-level</td></tr>
<tr><td colspan="2">Ref.: ITU-T Rec. Q.2965.1</td></tr>
<tr><td>AALtype</td><td>401A</td><td>1 octet</td><td colspan="2">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</td></tr>
</table>

C.5 Frame Relay

PropertyID	Property tag	Type	Value
DLCI	5001	Unsigned integer	Data link connection id
CID	5002	Unsigned integer	subchannel 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)

C.7 ATM AAL2

PropertyID	Property tag	Type	Value
AESA	7001	20 octets	AAL2 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	AAL2 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.

PropertyID	Property tag	Type	Value
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 AAL1

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	AAL1 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

PropertyID	Property tag	Type	Value
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
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 spare 0 0 0 0 1 1 1 1 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

PropertyID	Property tag	Type	Value
			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 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

PropertyID	Property tag	Type	Value
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
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.721 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 Rec. 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 — ITU-T Rec. X.263 | 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 (IETF RFC 791) (ITU-T Rec. X.263 | ISO/IEC TR 9577) 1 1 0 0 1 1 1 1 Point-to-point Protocol (IETF 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/H.248.1 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 AAL5 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

PropertyID	Property tag	Type	Value
SDP_V	B001	String	Protocol Version Ref.: IETF RFC 2327
SDP_O	B002	String	Owner/creator and session ID Ref.: IETF RFC 2327
SDP_S	B003	String	Session name Ref.: IETF RFC 2327
SDP_I	B004	String	Session identifier Ref.: IETF RFC 2327
SDP_U	B005	String	URI of descriptor Ref.: IETF RFC 2327
SDC_E	B006	String	email address Ref.: IETF RFC 2327
SDP_P	B007	String	phone number Ref.: IETF RFC 2327
SDP_C	B008	String	Connection information Ref.: IETF RFC 2327
SDP_B	B009	String	Bandwidth Information Ref.: IETF RFC 2327
SDP_Z	B00A	String	Time zone adjustment Ref.: IETF RFC 2327
SDP_K	B00B	String	Encryption Key Ref.: IETF RFC 2327
SDP_A	B00C	String	Zero or more session attributes Ref.: IETF RFC 2327

PropertyID	Property tag	Type	Value
SDP_T	B00D	String	Active Session Time Ref.: IETF RFC 2327
SDP_R	B00E	String	Zero or more repeat times Reference: IETF RFC 2327
SDP_M	B00F	String	Media type, port, transport and format Ref.: IETF 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

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, being 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 transaction identifier of each incoming message is compared to the transaction identifiers 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 transaction-id 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

Transaction identifiers 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 transaction identifiers, or pre-allocating non-overlapping ranges of identifiers to different processes. The implementations shall guarantee that unique transaction identifiers are allocated to all transactions that originate from a logical MGC (identical mId). MGs can simply detect duplicate transactions by looking at the transaction identifier and mId only.

D.1.2.2 Three-way handshake

The TransactionResponse Acknowledgement parameter can be found in any message. It carries a set of "confirmed transaction-id ranges". Entities may choose to delete the copies of the responses to transactions whose id is included in "confirmed transaction-id ranges" received in the transaction response messages. They should silently discard further commands when the transaction-id falls within these ranges.

The "confirmed transaction-id 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 transaction-id.

Messages that carry the "Transaction Response Acknowledgement" parameter may be transmitted in any order. The entity shall retain the "confirmed transaction-id ranges" receivedfor 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, "Transaction Pending". They SHOULD send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a Transaction Pending shall switch to a different repetition timer for repeating requests. The root Termination has a property (ProvisionalResponseTimerValue), which can be set to the requested maximum number of milliseconds between receipt of a command and transmission of the TransactionPending response. 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 Transaction Pending 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 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 become 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 very 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 repetition 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 is 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 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 IETF 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, being 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, "Transaction Pending". They should send this response if they receive a repetition of a transaction that is still being executed.

Entities that receive a Transaction Pending 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 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

PackageID: g (0x0001)

Version: 1

Extends: None

Description:

Generic package for commonly encountered items

E.1.1 Properties

None

E.1.2 Events

Cause

EventID: cause (0x0001)

Generic error event

EventsDescriptor parameters: None

ObservedEvents Descriptor Parameters:

General Cause

ParameterID: Generalcause (0x0001)

This parameter groups the failures into six groups, which the MGC may act upon.

Type: enumeration

Possible values:

"NR" Normal Release (0x0001)

"UR" Unavailable Resources (0x0002)

"FT" Failure, Temporary (0x0003)

"FP" Failure, Permanent (0x0004)

"IW" Interworking Error (0x0005)

"UN" Unsupported (0x0006)

Failure Cause

ParameterID: Failurecause (0x0002)

Possible values: OCTET STRING

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.

Signal Completion

EventID: sc (0x0002)

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, 7.1.17 and 7.2.7.

EventsDescriptor parameters: None

ObservedEvents Descriptor parameters:

Signal Identity

ParameterID: SigID (0x0001)

This parameter identifies the signal which has terminated. For a signal that is contained in a signal list, the signal list identity parameter should also be returned indicating the appropriate list.

Type: Binary: octet (string), Text: string

Possible values: a signal which has terminated. A signal shall be identified using the pkgdName syntax without wildcarding.

Termination Method

ParameterID: Meth (0x0002)

Indicates the means by which the signal terminated.

Type: enumeration

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

Signal List ID

ParameterID: SLID (0x0003)

Indicates to which signal list a signal belongs. The SignalList ID is only returned in cases where the signal resides in a signal list.

Type: integer

Possible values: any integer

E.1.3 Signals

None.

E.1.4 Statistics

None.

E.2 Base Root Package

Base Root Package

PackageID: root (0x0002)

Version: 1

Extends: None

Description:

This package defines Gateway wide properties.

E.2.1 Properties

MaxNrOfContexts

PropertyID: maxNumberOfContexts (0x0001)

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

Defined in: TerminationState

Characteristics: read only

MaxTerminationsPerContext

PropertyID: maxTerminationsPerContext (0x0002)

The maximum number of allowed terminations in a context, see 6.1

Type: integer

Possible values: any integer

Defined in: TerminationState

Characteristics: read only

normalMGExecutionTime

PropertyId: normalMGExecutionTime (0x0003)

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 integer, represents milliseconds

Defined in: TerminationState

Characteristics: read/write

normalMGCExecutionTime

PropertyId: normalMGCExecutionTime (0x0004)

Settable by the MGC to indicate the interval within which the MG should expects a response to any transaction from the MGC (exclusive of network delay)

Type: integer

Possible values: any integer, represents milliseconds

Defined in: TerminationState

Characteristics: read/write

MGProvisionalResponseTimerValue

PropertyId: MGProvisionalResponseTimerValue (0x0005)

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 integer, represents milliseconds

Defined in: TerminationState

Characteristics: read/write

MGCProvisionalResponseTimerValue

PropertyId: MGCProvisionalResponseTimerValue (0x0006)

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 integer, represents milliseconds

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

PackageID: tonegen (0x0003)

Version: 1

Extends: None

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 playtones. 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.

Designed to be extended only: Yes

E.3.1 Properties

None.

E.3.2 Events

None.

E.3.3 Signals

Play tone

SignalID: pt (0x0001)

Plays audio tone over an audio channel

Signal Type: Brief

Duration: Provisioned

Additional parameters:

Tone id list

ParameterID: tl (0x0001)

Type: list of tone ids

List of tones to be played in sequence. The list SHALL contain one or more tone ids.

Inter signal duration

ParameterID: ind (0x0002)

Type: integer

Timeout between two consecutive tones in milliseconds

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.3.4 Statistics

None.

E.3.5 Procedures

None.

E.4 Tone Detection Package

PackageID: tonedet (0x0004)

Version: 1

Designed to be extended only: Yes

Extends: None

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.

E.4.1 Properties

None.

E.4.2 Events

Start tone detected

EventID: std, 0x0001

Detects the start of a tone. The characteristics of positive tone detection are implementation dependent.

EventsDescriptor parameters:

Tone id list

ParameterID: tl (0x0001)

Type: list of tone ids

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 "wild card", any tone id is detected.

ObservedEventsDescriptor parameters:

Tone id

ParameterID: tid (0x0003)

Type: enumeration

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.

End tone detected

EventID: etd, 0x0002

Detects the end of a tone.

EventDescriptor parameters:

Tone id list

ParameterID: tl (0x0001)

Type: enumeration or list of enumerated types

Possible values: No possible values are specified in this package. Extensions to this package would add possible values for tone id.

ObservedEventsDescriptor parameters:

Tone id

ParameterID: tid (0x0003)

Type: enumeration

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.

Duration

ParameterId: dur (0x0002)

Type: integer, in milliseconds

This parameter contains the duration of the tone from first detection until it stopped.

Long tone detected

EventID: ltd, 0x0003

Detects that a tone has been playing for at least a certain amount of time

EventDescriptor parameters:

Tone id list

ParameterID: tl (0x0001)

Type: enumeration or list

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.

Duration

ParameterID: dur (0x0002)

Type: integer, duration to test against

Possible values: any legal integer, expressed in milliseconds

ObservedEventsDescriptor parameters:

Tone id

ParameterID: tid (0x0003)

Type: Enumeration

Possible values: No possible values are specified in this package. Extensions to this package would add possible values for tone id.

E.4.3 Signals

None.

E.4.4 Statistics

None.

E.4.5 Procedures

None.

E.5 Basic DTMF Generator Package

PackageID: dg (0x0005)

Version: 1

Extends: tonegen version 1

This package defines the basic DTMF tones as signals and extends the allowed values of parameter tl of playtone in tonegen.

E.5.1 Properties

None.

E.5.2 Events

None.

E.5.3 Signals

DTMF character 0

SignalID: d0 (0x0010)

Generate DTMF 0 tone. The physical characteristic of DTMF 0 is defined in the gateway.

Signal Type: Brief

Duration: Provisioned

Additional parameters:

None

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 signal IDs 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 digit map.

Signal name	Signal ID/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)
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

PackageID: dd (0x0006)

Version: 1

Extends: tonedet version 1

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 digit map 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"

E.6.1 Properties

None.

E.6.2 Events

DTMF digits

EventIds are defined with the same names as the SignalIds defined in the table found in E.5.3.

DigitMap Completion Event

EventID: ce, 0x0004

Generated when a digit map completes as described in 7.1.14.

EventsDescriptor parameters: None.

ObservedEventsDescriptor parameters:

DigitString

ParameterID: ds (0x0001)

Type: string of digit map symbols (possibly empty) returned as a quotedString

Possible values: a sequence of the characters "0" through "9", "A" through "F", and the long duration modifier "Z".

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 digit map.

Termination Method

ParameterID: Meth (0x0003)

Type: enumeration

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

Description: indicates the reason for generation of the event. See the procedures in 7.1.14.

E.6.3 Signals

None.

E.6.4 Statistics

None.

E.6.5 Procedures

Digit map processing is activated only if an events descriptor is activated that contains a digit map completion event as defined in E.6.2 and that digit map completion event contains an eventDM field in the requested actions as defined in 7.1.9. Other parameters, such as KeepActive or embedded events of signals descriptors, may also be present in the events descriptor and do not affect the activation of digit map processing.

E.7 Call Progress Tones Generator Package

PackageID: cg, 0x0007

Version: 1

Extends: tonegen version 1

This package defines the basic call progress tones as signals and extends the allowed values of the tl parameter of playtone in tonegen.

E.7.1 Properties

None.

E.7.2 Events

None.

E.7.3 Signals

Dial Tone

SignalID: dt (0x0030)

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 signal IDs 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	Signal ID/tone id
Dial Tone	dt (0x0030)
Ringing Tone	rt (0x0031)
Busy Tone	bt (0x0032)
Congestion Tone	ct (0x0033)
Special Information Tone	sit (0x0034)
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

PackageID: cd (0x0008)

Version: 1

Extends: tonedet version 1

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.

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

PackageID: al, 0x0009

Version: 1

Extends: None

This package defines events and signals for an analog line.

E.9.1 Properties

None

E.9.2 Events

onhook

EventID: on (0x0004)

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.

EventDescriptor parameters:

Strict Transition

ParameterID: strict (0x0001)

Type: enumeration

Possible values: "exact" (0x00), "state" (0x01), "failWrong" (0x02)

"exact" means that only an actual hook state transition to on-hook is to be recognized;

"state" means that the event is to be recognized either if the hook state transition is detected or if the hook state is already on-hook;

"failWrong" means that if the hook state is already on-hook, the command fails and an error is reported.

ObservedEventsDescriptor parameters:

Initial State

ParameterID: init (0x0002)

Type: Boolean

Possible values:

"True" means that the event was reported because the line was already on-hook when the events descriptor containing this event was activated;

"False" means that the event represents an actual state transition to on-hook.

offhook

EventID: of (0x0005)

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.

EventDescriptor parameters:

Strict Transition

ParameterID: strict (0x0001)

Type: enumeration

Possible values: "exact" (0x00), "state" (0x01), "failWrong" (0x02)

"exact" means that only an actual hook state transition to off-hook is to be recognized;

"state" means that the event is to be recognized either if the hook state transition is detected or if the hook state is already off-hook;

"failWrong" means that if the hook state is already off-hook, the command fails and an error is reported.

ObservedEventsDescriptor parameters:

Initial State

ParameterID: init (0x0002)

Type: Boolean

Possible values:

"True" means that the event was reported because the line was already off-hook when the events descriptor containing this event was activated;

"False" means that the event represents an actual state transition to off-hook.

flashhook

EventID: fl, 0x0006

Detects handset flash. A flash occurs when an onhook is followed by an offhook between a minimum and maximum duration.

EventDescriptor parameters:

Minimum duration

ParameterID: mindur (0x0004)

Type: integer in milliseconds

Default value is provisioned.

Maximum duration

ParameterID: maxdur (0x0005)

Type: integer in milliseconds

Default value is provisioned.

ObservedEventsDescriptor parameters:

None

E.9.3 Signals

ring

SignalID: ri, 0x0002

Applies ringing on the line

Signal Type: TimeOut

Duration: Provisioned

Additional parameters:

Cadence

ParameterID: cad (0x0006)

Type: list of integers representing durations of alternating on and off segments, constituting a complete ringing cycle starting with an on. Units in milliseconds

Default is fixed or provisioned. Restricted function MGs may ignore cadence values they are incapable of generating.

Frequency

ParameterID: freq (0x0007)

Type: integer in Hz

Default is fixed or provisioned. Restricted function MGs may ignore frequency values they are incapable of generating.

E.9.4 Statistics

None

E.9.5 Procedures

If the MGC sets an EventsDescriptor 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 EventsDescriptor fails. The MG SHALL include error code 540 (Unexpected initial hook state) in its reponse.

E.9.6 Error code

This package defines a new error code:

540 (Unexpected initial hook state)

The procedure for use of this code is given in E.9.5.

E.10 Basic Continuity Package

PackageID: ct (0x000a)

Version: 1

Extends: None

This package defines events and signals for continuity test. The continuity test includes provision of either a loopback or transceiver functionality.

E.10.1 Properties

None.

E.10.2 Events

Completion

EventID: cmp, 0x0005

This event detects test completion of continuity test.

EventDescriptor parameters:

None

ObservedEventsDescriptor parameters:

Result

ParameterID: res (0x0008)

Type: enumeration

Possible values: success (0x0001), failure (0x0000)

E.10.3 Signals

Continuity test

SignalID: ct (0x0003)

Initiates sending of continuity test tone on the termination to which it is applied.

Signal Type: TimeOut

Default value is provisioned

Additional parameters:

None

Respond

SignalID: rsp (0x0004)

The signal is used to respond to a continuity test. See E.10.5 for further explanation.

Signal Type: On/Off

Default duration is 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.

(Informative 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

PackageID: nt (0x000b)

Version: 1

Extends: None

This package defines properties of network terminations independent of network type.

E.11.1 Properties

Maximum Jitter Buffer

PropertyID: jit (0x0007)

This property puts a maximum size on the jitter buffer.

Type: integer in milliseconds

Possible values: This property is specified in milliseconds.

Defined in: LocalControlDescriptor

Characteristics: read/write

E.11.2 Events

network failure

EventID: netfail, 0x0005

The termination generates this event upon detection of a failure due to external or internal network reasons.

EventDescriptor parameters:

None

ObservedEventsDescriptor parameters:

cause

ParameterID: cs (0x0001)

Type: string

Possible values: any text string

This parameter may be included with the failure event to provide diagnostic information on the reason of failure.

quality alert

EventID: qualert, 0x0006

This property allows the MG to indicate a loss of quality of the network connection. The MG may do this by measuring packet loss, interarrival jitter, propogation delay and then indicating this using a percentage of quality loss.

EventDescriptor parameters:

Threshold

ParameterId: th (0x0001)

Type: integer

Possible values: 0 to 99

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.

ObservedEventsDescriptor parameters:

Threshold

ParameterId: th (0x0001)

Type: integer

Possible values: 0 to 99

Description: percent of quality loss measured, calculated based on a provisioned method, that could take into consideration packet loss, jitter, and delay for example.

E.11.3 Signals

None.

E.11.4 Statistics

Duration

StatisticsID: dur (0x0001)

Description: provides duration of time the termination has been in the Context.

Type: double, in milliseconds

Octets Sent

StatisticID: os (0x0002)

Type: double

Possible values: any 64-bit integer

Octets Received

StatisticID: or (0x0003)

Type: double

Possible values: any 64-bit integer

E.11.5 Procedures

None.

E.12 RTP Package

PackageID: rtp (0x000c)

Version: 1

Extends: Network Package version 1

This package is used to support packet-based multimedia data transfer by means of the Real-time Transport Protocol (RTP) (IETF RFC 1889).

E.12.1 Properties

None.

E.12.2 Events

Payload Transition

EventID: pltrans, 0x0001

This event detects and notifies when there is a transition of the RTP payload format from one format to another.

EventDescriptor parameters:

None

ObservedEventsDescriptor parameters:

ParameterName: rtppayload

ParameterID: rtppltype, 0x01

Type: list of enumerated types.

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.

E.12.3 Signals

None.

E.12.4 Statistics

Packets Sent

StatisticID: ps (0x0004)

Type: double

Possible values: any 64-bit integer

Packets Received

StatisticID: pr (0x0005)

Type: double

Possible values: any 64-bit integer

Packet Loss

StatisticID: pl (0x0006)

Describes the current rate of packet loss on an RTP stream, as defined in IETF RFC 1889. 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.

Jitter

StatisticID: jit (0x0007)

Requests the current value of the interarrival jitter on an RTP stream as defined in IETF RFC 1889. Jitter measures the variation in interarrival time for RTP data packets.

Delay

StatisticID:delay (0x0008)

Requests the current value of packet propagation delay expressed in timestamp units. Same as average latency.

E.12.5 Procedures

None.

E.13 TDM Circuit Package

PackageID: tdmc (0x000d)

Version: 1

Extends: Network Package version 1

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.

E.13.1 Properties

Echo Cancellation

PropertyID: ec (0x0008)

Type: boolean

Possible values:

"on" (when the echo cancellation is requested) and

"off" (when it is turned off.)

The default is provisioned.

Defined in: LocalControlDescriptor

Characteristics: read/write

Gain Control

PropertyID: gain (0x000a)

Gain control, or usage of of signal level adaptation and noise level reduction is used to adapt the level of the signal. However, it is necessary, for example for modem calls, to turn off this function.

Type: integer

Possible values:

The gain control parameter may either be specified as "automatic" (0xffffffff), or as an explicit number of decibels of gain (any other integer value). The default is provisioned in the MG.

Defined in: LocalControlDescriptor

Characteristics: read/write

E.13.2 Events

None.

E.13.3 Signals

None.

E.13.4 Statistics

None.

E.13.5 Procedures

None.

Appendix I

Example Call Flows

All H.248.1 implementors must read the normative part of this Recommendation carefully before implementing from it. The examples in this appendix should not be used as stand-alone explanations of how to create protocol messages.

The examples in this appendix use SDP for encoding of the Local and Remote stream descriptors. SDP is defined in IETF RFC 2327. If there is any discrepancy between the SDP in the examples, and IETF RFC 2327, the RFC should be consulted for correctness. Audio profiles used are those defined in IETF 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 AnalogLine 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 default 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,
           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. Mode for this stream is set to SendReceive. "al" is the analog line supervision package.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Transaction = 9999 {
   Context = - {
       Modify = A4444 {
           Media { Stream = 1 {
                    LocalControl {
                        Mode = SendReceive,
                        tdmc/gain=2,  ; in dB,
                        tdmc/ec=on
                    },
                    Local {
v=0
c=IN IP4 $
m=audio $ RTP/AVP 0
a=fmtp:PCMU VAD=X-NNVAD ; special voice activity
                        ; detection algorithm
                    }
                }
           },
```

```
            Events = 2222 {al/of}
        }
    }
}
```

The dialplan script could have been loaded into the MG previously. Its function would be to wait for the OffHook, turn on dialtone and start collecting DTMF digits. However, in this example, we use the digit map, which is put into place after the offhook is detected [step 5) below].

Note that the embedded EventsDescriptor 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/1 [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 offhook event from User 1 and reports it to the Media Gateway Controller via the Notify Command.

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Transaction = 10000 {
  Context = - {
      Notify = A4444 {ObservedEvents =2222 {
        19990729T22000000:al/of}}
  }
}
```

7) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/1 [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/1 [123.123.123.4]:55555
Transaction = 10001 {
   Context = - {
       Modify = A4444 {
           Events = 2223 {
               al/on, dd/ce {DigitMap=Dialplan0}
           },
```

```
           Signals {cg/dt},
           DigitMap= Dialplan0{
(0| 00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxx|9011x.)}
        }
   }
}
```

9) And the Modify is acknowledged.

```
MG1 to MGC:
MEGACO/1 [124.124.124.222]:55555
Reply = 10001 {
   Context = - {Modify = A4444}
}
```

10) Next, digits are accumulated by MG1 as they are dialed 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/1 [124.124.124.222]:55555
Transaction = 10002 {
  Context = - {
      Notify = A4444 {ObservedEvents =2223 {
        19990729T22010001:dd/ce{ds="916135551212",Meth=FM}}}
  }
}
```

11) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/1 [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 ReceiveOnly since Remote descriptor values are not yet specified. Preferred codecs are in the MGC's preferred order of choice.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Transaction = 10003 {
   Context = $ {
      Add = A4444,
      Add = $ {
          Media {
            Stream = 1 {
                 LocalControl {
                     Mode = ReceiveOnly,

                     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/1 [124.124.124.222]:55555
Reply = 10003 {
  Context = 2000 {
     Add = A4444,
     Add=A4445{
        Media {
            Stream = 1 {
                Local {
v=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), SendReceive connection through to the originating user, User 1. The MGC also sets ring on A5555.

```
MGC to MG2:
MEGACO/1 [123.123.123.4]:55555
Transaction = 50003 {
   Context = $ {
      Add = A5555  { Media {
           Stream = 1 {
                LocalControl {Mode = SendReceive} }},
       Events=1234{al/of},
           Signals {al/ri}
           },
      Add  = $ {Media {
           Stream = 1 {
                LocalControl {
                   Mode = SendReceive,
                   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/1 [124.124.124.222]:55555
Reply = 50003 {
  Context = 5000 {
     Add = A5555,
     Add = A5556{
        Media {
           Stream = 1 {
               Local {
v=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/1 [123.123.123.4]:55555
Transaction = 10005 {
 Context = 2000 {
   Modify = A4444 {
     Signals {cg/rt}
   },
   Modify = A4445 {
      Media {
           Stream = 1 {
               Remote {
v=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/1 [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/1 [125.125.125.111]:55555
Transaction = 50005 {
  Context = 5000 {
      Notify = A5555 {ObservedEvents =1234 {
        19990729T22020002:al/of}}
  }
}

From MGC to MG2:

MEGACO/1 [123.123.123.4]:55555
Reply = 50005 {
   Context = - {Notify = A5555}
}

From MGC to MG2:

MEGACO/1 [123.123.123.4]:55555
Transaction = 50006 {
  Context = 5000 {
     Modify = A5555 {
        Events = 1235 {al/on},
         Signals { } ; to turn off ringing
     }
  }
}

From MG2 to MGC:

MEGACO/1 [125.125.125.111]:55555
Reply = 50006 {
 Context = 5000 {Modify = A4445}
}
```

18) Change mode on MG1 to SendReceive, and stop the ringback.

```
MGC to MG1:
MEGACO/1 [123.123.123.4]:55555
Transaction = 10006 {
  Context = 2000 {
     Modify = A4445 {
        Media {
           Stream = 1 {
              LocalControl {
                 Mode=SendReceive
              }
           }
        }
     },
     Modify = A4444 {
        Signals { }
     }
  }
}

from MG1 to MGC:
MEGACO/1 [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/1 [123.123.123.4]:55555
Transaction = 50007 {
  Context = - {AuditValue = A5556{
     Audit{Media, DigitMap, Events, Signals, Packages, Statistics }}
  }
}
```

20) The MG2 replies.

```
MEGACO/1 [125.125.125.111]:55555
Reply = 50007 {
  Context = - {
AuditValue = A5556 {
         Media {
            TerminationState { ServiceStates = InService,
          Buffer = OFF },
        Stream = 1 {
                LocalControl { Mode = SendReceive,
                   nt/jit=40 },
                Local {
v=0
c=IN IP4 125.125.125.111
m=audio 1111 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
                } } },
        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 onhook 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/1 [125.125.125.111]:55555
Transaction = 50008 {
  Context = 5000 {
      Notify = A5555 {ObservedEvents =1235 {
         19990729T24020002:al/on}
      }
  }
}
```

```
From MGC to MG2:

MEGACO/1 [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 subtracts 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/1 [123.123.123.4]:55555
Transaction = 50009 {
  Context = 5000 {
     Subtract = A5555 {Audit{Statistics}},
     Subtract = A5556 {Audit{Statistics}}
  }
}

From MG2 to MGC:

MEGACO/1 [125.125.125.111]:55555
Reply = 50009 {
  Context = 5000 {
    Subtract = A5555 {
         Statistics {
            nt/os=45123, ; Octets Sent
            nt/dur=40 ; in seconds
            }
      },
      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
         }
      }
  }
}
```

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.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
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	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
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 and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2002