INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.248.1 v2

Corrigendum 1

(03/2004)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Communication procedures

Gateway control protocol: Version 2

Corrigendum 1

ITU-T Recommendation H.248.1 v2 (2002) –
Corrigendum 1

ITU-T H-SERIES RECOMMENDATIONS

AUDIOVISUAL AND MULTIMEDIA SYSTEMS

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

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

ITU-T Recommendation H.248.1 v2

Gateway control protocol: Version 2

Corrigendum 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 H.323.

This Recommendation includes several enhancements to ITU-T Rec. H.248.1 Version 1:

– individual property, signal, event and statistic auditing;

– improved multiplex handling;

– topology for streams;

– improved description of profiles;

– serviceChange capability change.

Corrigendum 1 corrects several defects found in the Recommendation, in particular:

- Specification of types for rtp/jit and rtp/delay in E.12.4;
- Definition of the '#' symbol in the "unequal" construct in text encoding;
- Definition of the symbol for the NULL context in text encoding;
- Corrections: to Appendix I example statistics, package guidelines for statistics in 12.1.5; ambiguous audit and individual audit return; Context Audit Return; Typographical error in 7.1.2;
- Specification of the meaning of "automatic" in E.13 "TDM Package";
- Additional codepoint and of binary value for packetization time in Annex C;
- Clarification of wildcarding principles and of wildcarding in the Topology Descriptor;
- Clarification of: statistics and the Move command; modification of terminations by MGCs; optional commands in an action; ordering of transactions; precedence of LocalControl mode property versus SDP mode; Digit processing; usage of digitmap timer symbols with range notation; use of StreamID = 0; AuditCapabilities return for string values; protocol version negotiation;
- Clarification that the network package can apply to TDM.

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

Source

Corrigendum 1 to ITU-T Recommendation H.248.1 v2 (2002) was approved on 15 March 2004 by ITU-T Study Group 16 (2001-2004) under the ITU-T Recommendation A.8 procedure.

CONTENTS

ITU-T Recommendation H.248.1 v2

Gateway control protocol: Version 2

Corrigendum 1

1) Clause 2.1

Modify 2.1 as follows:

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.248.4 (2000), *Gateway control protocol: Transport over Stream Control Transmission Protocol (SCTP)*, plus Cor.1 (2004).

– ITU-T Recommendation H.248.5 (2000), *Gateway control protocol: Transport over ATM*.

– ITU-T Recommendation H.248.8 (2002), *Gateway control protocol: Error code and service change reason description*, plus Amendment 1 (2004).

•••••

2) Clause 6.2

Modify 6.2 as follows:

6.2 Terminations

•••••

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 subtracted from a context~~ceases to exist or is returned to the NULL context due to a Subtract command.

•••••

6.2.1 Termination dynamics

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

6.2.2 TerminationIDs

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

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

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

~~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.

•••••

6.2.4 Termination properties and descriptors

•••••

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

Descriptor name	Description
Modem	Identifies modem type and properties when applicable. (Note)

•••••

Error	Contains an error code and optionally error text; it may occur in command replies and in Notify requests.
NOTE – ModemDescriptor has been deprecated in ITU-T Rec. H.248.1 version ~~1~~2 (0~~3~~5/2002).	

6.2.5 Root termination

•••••

3) Clause 6.3

Add 6.3 as follows:

6.3 Wildcarding principles

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

1) Context Wildcarding;

2) Termination Wildcarding.

When executing a transaction that contains wildcarded contexts and optionally wildcarded terminations, all commands in the transaction are executed in order for a particular instance of ContextID before moving to a subsequent ContextID instance. In the case that there are multiple commands in a transaction, only when the TerminationID (wildcarded or specific) specified in the first command matches a specific instance of a ContextID are subsequent commands in the transaction executed. If a TerminationID (wildcarded or specific) of the subsequent command(s) in that transaction does not match the specific ContextID instance, then an error code 431 is returned and processing of subsequent instances of the wildcard ContextID is stopped unless the command that generated the error is marked optional.

The execution of particular wildcard combinations is discussed below.

6.3.1 ContextID specific with TerminationID wildcarded

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

For example: Assume that a gateway has 4 terminations: t1/1, t1/2, t2/1 and t2/2. Assume that Context 1 has t1/1 and t2/1 in it and that Context 2 has t1/2 and t2/2 in it.

The command:

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

Returns:

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

6.3.2 ContextID wildcarded (ALL) with TerminationID specific

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

For example: Taking the above gateway configuration.

The command:

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

Returns:

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

6.3.3 ContextID wildcarded (ALL) with TerminationID wildcarded

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

For example: Taking the above gateway configuration.

The command:

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

Returns:

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

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

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

6.3.4 Wildcarded responses

If individual responses are not required, a wildcard response may be requested. In such a case, a single response is generated, which contains the UNION of all of the individual responses which otherwise would have been generated, with duplicate values suppressed. For instance, given 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. If a wildcard UNION response is used in conjunction with a wildcarded Context, then a single response is sent with the UNION of all the individual termination/s referenced by the TerminationID. The response would contain Context = ALL, a wildcarded TerminationID and the sequence of properties.

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

For example

The command:

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

Response to an error:

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

Command=t1/3{Error=errorcode}}

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

•••••

4) Clause 7, Commands

a) Modify 7.1 as follows:

7.1.2 Modem descriptor

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

Use of the ModemDescriptor is deprecated in ITU-T Rec. H.248.1 version ~~1 (03~~2 (05/2002) and subsequent versions. This means that the ModemDescriptor shall not be included as part of transmitted content and, if received, shall either be ignored or processed at the option of the implementation. Modem type is to be specified as an attribute of data streams in LocalDescriptor and RemoteDescriptor.

•••••

7.1.4 Media descriptor

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

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

•••••

7.1.7 LocalControl descriptor

The LocalControl descriptor contains the Mode property, the ReserveGroup and ReserveValue properties and properties of a Termination (defined in Packages) that are stream specific, and are of interest between the MG and the MGC. Values of properties may be specified as in 7.1.1.

The allowed values for the mode property are 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 LocalControl Mode property takes precedence over any mode specified in the Local and Remote descriptors.

•••••

7.1.14.2 DigitMap timers

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

1) The start timer (T) is used prior to any digits being available for processing against the digit map~~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.

•••••

7.1.14.3 DigitMap syntax

•••••

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

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

7.1.14.4 DigitMap completion event

•••••

7.1.15 Statistics descriptor

The Statistics Descriptor provides information describing the status and usage of a Termination during its existence (ephemeral) or while it is outside the NULL context (physical)~~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 ceases to exist or is returned to the NULL context due to a Subtract command~~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 ceases to exist or is returned to the NULL context due to a Subtract command~~is Subtracted from a Context~~.

7.1.16 Packages descriptor

•••••

7.1.18 Topology descriptor

•••••

- (*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~~ either *T1* ~~and~~ or *T2* *but not both* 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.

•••••

b) *Modify 7.2 as follows:*

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]
)
```

(*) ModemDescriptor has been deprecated in H.248.1 version ~~1~~2 (05~~3~~/2002).

•••••

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]
)
```

(*) ModemDescriptor has been deprecated in H.248.1 version ~~1~~2 (0~~3~~5/2002).

•••••

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]
          )
```

(*) ModemDescriptor has been deprecated in H.248.1 version ~~1~~2 (0~~3~~5/2002).

•••••

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]
          )
```

(*) ModemDescriptor has been deprecated in H.248.1 version 2~~1~~ (05~~3~~/2002).

•••••

7.2.5 AuditValue

The AuditValue Command returns the current values of properties, events, signals and statistics associated with Terminations. An AuditValue may request the contents of a descriptor or of a single property, event, signal or statistics.

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor] (*)
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,DigitMapDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
[,PackagesDescriptor]
        AuditValue(TerminationID,
            AuditDescriptor
        )
```

(*) ModemDescriptor has been deprecated in H.248.1 version 2~~1~~ (05~~3~~/2002).

•••••

7.2.6 AuditCapabilities

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

```
TerminationID
[,MediaDescriptor]
[,ModemDescriptor](*)
[,MuxDescriptor]
[,EventsDescriptor]
[,SignalsDescriptor]
[,ObservedEventsDescriptor]
[,EventBufferDescriptor]
[,StatisticsDescriptor]
        AuditCapabilities(TerminationID,
            AuditDescriptor)
```

(*) ModemDescriptor has been deprecated in H.248.1 version 2~~1~~ (05~~3~~/2002).

•••••

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.

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

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

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

The 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.

•••••

5) Clause 8

Modify clause 8 as follows:

8 Transactions

•••••

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

Transactions guarantee ordered Command processing. That is, Commands within a Transaction are executed sequentially. Ordering of Transactions is NOT guaranteed; transactions may be executed in any order, or simultaneously; however, transaction replies should be executed before transaction requests when both are contained in a message.

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

•••••

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.

•••••

6) Clause 11.3

Modify 11.3 as follows:

11.3 Negotiation of protocol version

A ServiceChange command from a MG that registers with an MGC shall contain the version number of the protocol supported by the MG in the ServiceChangeVersion parameter. Regardless of the version placed in the ServiceChangeVersion parameter, the message containing the command shall be encoded as a version 1 message. Upon receiving such a message, if the MGC supports only a lower version, then the MGC shall send a 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 only supports higher version(s) than the MG, it shall reject the association with Error 406 (Version Not Supported).

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

•••••

7) Clause 12

Modify clause 12 as follows:

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:

Type: One of:

Boolean

String: UTF-8 string

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

Integer: 4 byte signed integer

Double: 8 byte signed integer

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

Enumeration: One of a list of possible unique values (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: sub-list of enumeration. The encoding of sub-lists is specified in Annex A and B.3.

Possible Values:

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

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

12.1.6 Procedures

•••••

12.5 Package registration

A package can be registered with IANA for interoperability reasons. See clause ~~13~~ 14 for IANA considerations.

•••••

8) Annex B Text encoding of the protocol

Modify Annex B as follows:

•••••

B.2 ABNF specification

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

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

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

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

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

megacoMessage        = LWSP [authenticationHeader SEP ] message

authenticationHeader = AuthToken EQUAL SecurityParmIndex COLON
                       SequenceNum COLON AuthData

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

message            = MegacopToken SLASH Version SEP mId SEP messageBody
; The version of the protocol defined here is equal to 2.

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

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 / auditReturnItem)

auditReturnItem       = (MuxToken / ModemToken / MediaToken /
                         DigitMapToken / StatsToken /
                         ObservedEventsToken / PackagesToken)

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.
;'-' is used for NULL context.
ContextID             = (UINT32 / "*" / "-" / "$")

domainAddress         = "[" (IPv4address / IPv6address) "]"
;RFC 2373 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 ; '#' means "not equal"
LSBRKT               = LWSP "[" LWSP
RSBRKT               = LWSP "]" LWSP

; Note - The octet zero is not among the permitted characters in octet
; string. As the current definition is limited to SDP, and a zero octet
; would not be a legal character in SDP, this is not a concern.
localDescriptor      = LocalToken LBRKT octetString RBRKT

remoteDescriptor     = RemoteToken LBRKT octetString RBRKT

eventBufferDescriptor= EventBufferToken [ LBRKT eventSpec
                       *( COMMA eventSpec) RBRKT ]

eventSpec      = pkgdName [ LBRKT eventSpecParameter
                 *(COMMA eventSpecParameter) RBRKT ]
eventSpecParameter   = (eventStream / eventOther)

eventBufferControl     = BufferToken EQUAL ( "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 / Nx64kToken )

StreamID             = UINT16
pkgdName             = (PackageName SLASH ItemID) ;specific item
                            / (PackageName SLASH "*") ;all items in package
                            / ("*" SLASH "*") ; all items supported by the MG
PackageName          = NAME
ItemID               = NAME

eventsDescriptor     = EventsToken [ EQUAL RequestID LBRKT
                       requestedEvent *( COMMA requestedEvent ) RBRKT ]

requestedEvent       = pkgdName [ LBRKT eventParameter
                       *( COMMA eventParameter ) RBRKT ]

; 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] ["Z" COLON Timer COMMA]
                        digitMap
Timer                = 1*2DIGIT
; Units are seconds for T, S, and L timers, and hundreds of
; milliseconds for Z timer. Thus T, S, and L range from 1 to 99
; seconds and Z from 100 ms to 9.9 s
digitMap = (digitString / LWSP "(" LWSP digitStringList LWSP ")" LWSP)
digitStringList      = digitString *( LWSP "|" LWSP digitString )
digitString          = 1*(digitStringElement)
digitStringElement   = digitPosition [DOT]
digitPosition        = digitMapLetter / digitMapRange
digitMapRange        = ("x" / (LWSP "[" LWSP digitLetter LWSP "]" LWSP))
digitLetter          = *((DIGIT "-" DIGIT ) / digitMapLetter)
digitMapLetter       = DIGIT   ;Basic event symbols
                        / %x41-4B / %x61-6B ; a-k, A-K
                        / "L" / "S" / "T" ;Inter-event timers
                        ; (long, short, start)
                        / "Z"    ;Long duration modifier

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

indAudterminationAudit   = indAudauditReturnParameter
                            *(COMMA indAudauditReturnParameter)

indAudauditReturnParameter = (indAudmediaDescriptor / /
                              indAudeventsDescriptor /
                              indAudsignalsDescriptor /
                              indAuddigitMapDescriptor /
                              indAudeventBufferDescriptor /
                              indAudstatisticsDescriptor /
                              indAudpackagesDescriptor)


indAudmediaDescriptor   = MediaToken LBRKT indAudmediaParm RBRKT
```

```
; at-most-once per item
; and either streamParm or streamDescriptor but not both
indAudmediaParm            = (indAudstreamParm / indAudstreamDescriptor /
                                 indAudterminationStateDescriptor)

; at-most-once
indAudstreamParm           = ( indAudlocalControlDescriptor )
; SDP too complex to pull out individual pieces for audit,
; hence no individual audit for Local and Remote

indAudstreamDescriptor     = StreamToken EQUAL StreamID
                                  LBRKT indAudstreamParm RBRKT

indAudlocalControlDescriptor = LocalControlToken LBRKT indAudlocalParm RBRKT

; at-most-once per item
indAudlocalParm            = ( ModeToken / pkgdName /
                                ReservedValueToken /
                                ReservedGroupToken )


indAudterminationStateDescriptor = TerminationStateToken LBRKT
                              indAudterminationStateParm RBRKT

; at-most-once per item
indAudterminationStateParm =(pkgdName / ServiceStatesToken / BufferToken)

indAudeventBufferDescriptor = EventBufferToken LBRKT indAudeventSpec RBRKT

indAudeventSpec         = pkgdName [ LBRKT indAudeventSpecParameter RBRKT ]
indAudeventSpecParameter   = (eventStream / eventParameterName)

indAudeventsDescriptor     = EventsToken EQUAL RequestID LBRKT
                            indAudrequestedEvent RBRKT

indAudrequestedEvent       = pkgdName

indAudsignalsDescriptor    = SignalsToken LBRKT [ indAudsignalParm ] RBRKT

indAudsignalParm           = indAudsignalList / indAudsignalRequest

indAudsignalRequest        = signalName
indAudsignalList           = SignalListToken EQUAL signalListId LBRKT
                                 indAudsignalListParm RBRKT

indAudsignalListParm    = indAudsignalRequest

indAuddigitMapDescriptor   = DigitMapToken EQUAL (digitMapName )

indAudstatisticsDescriptor = StatsToken LBRKT pkgdName RBRKT

indAudpackagesDescriptor   = PackagesToken LBRKT packagesItem RBRKT


serviceChangeDescriptor = ServicesToken LBRKT serviceChangeParm
                         *(COMMA serviceChangeParm) RBRKT

; each parameter at-most-once, except auditItem
; at most one of either serviceChangeAddress or serviceChangeMgcId but
; not both
; serviceChangeMethod and serviceChangeReason are REQUIRED
serviceChangeParm    = (serviceChangeMethod / serviceChangeReason /
                       serviceChangeDelay / serviceChangeAddress /
                       serviceChangeProfile / extension / TimeStamp /
                       serviceChangeMgcId / serviceChangeVersion /
                       auditItem)

serviceChangeReplyDescriptor = ServicesToken LBRKT
                       servChgReplyParm *(COMMA servChgReplyParm) RBRKT

; at-most-once. Version is REQUIRED on first ServiceChange response
; at most one of either serviceChangeAddress or serviceChangeMgcId but
; not both
servChgReplyParm     = (serviceChangeAddress / serviceChangeMgcId /
                       serviceChangeProfile / serviceChangeVersion /
                       TimeStamp)
serviceChangeMethod  = MethodToken EQUAL (FailoverToken /
                       ForcedToken / GracefulToken / RestartToken /
                       DisconnectedToken / HandOffToken /
                       extensionParameter)
; A serviceChangeReason consists of a numeric reason code
; and an optional text description.
; A serviceChangeReason MUST be encoded using the quotedString
; form of VALUE.
; The quotedString SHALL contain a decimal reason code,
; optionally followed by a single space character and a
; textual description string.


serviceChangeReason  = ReasonToken  EQUAL VALUE
serviceChangeDelay   = DelayToken   EQUAL UINT32
serviceChangeAddress = ServiceChangeAddressToken EQUAL ( mId /
                       portNumber )
serviceChangeMgcId   = MgcIdToken   EQUAL mId
serviceChangeProfile = ProfileToken EQUAL NAME SLASH Version
serviceChangeVersion = VersionToken EQUAL Version
extension            = extensionParameter parmValue

packagesDescriptor   = PackagesToken LBRKT packagesItem
                       *(COMMA packagesItem) RBRKT

Version              = 1*2(DIGIT)
packagesItem         = NAME "-" UINT16

TimeStamp            = Date "T" Time ; per ISO 8601:1988
; 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
                       [COMMA eventStream ]
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")
```

```
EmergencyOffToken           = ("EmergencyOffToken"      / "EGO")
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")
Nx64kToken                  = ("Nx64Kservice"           / "N64")
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")
```

•••••

9) Annex C Tags for media stream properties

•••••

Modify Annex C as follows:

C.1 General media attributes

PropertyID	Property tag	Type	Value

•••••

Gain	100C	~~Unsigned integer~~	Not Used. See E.13 for an available gain property.~~Gain in dB: 0..65535~~

•••••

Ptime	1010	Integer	Packetization Time This gives the length of time in milliseconds represented by the media in a packet. Ref.: IETF RFC 2327

C.2 Mux properties

•••••

C.12 H.245

PropertyID	Property tag	Type	Value

•••••

CLCack	C006	Octet string	The value of H.245 CloseLogicalChannelAck structure. Ref.: ITU-T Rec. H.245
LCN	C007	Integer	The value of H.245 Local Channel Number 0-65535. Ref.: ITU-T Rec. H.245

Annex D

Transport over IP

•••••

10) Annex E Basic packages

•••••

Modify Annex E as follows:

E.11 Network Package

PackageID: nt (0x000b)

Version: 1

Extends: None

This package defines properties of network terminations independent of network type. This includes, but is not limited to, TDM, IP and ATM.

•••••

E.11.4 Statistics

Duration

StatisticsID: dur (0x0001)

Description: provides duration of time the termination has existed or been out of the null context~~been in the Context~~.

Type: double, in milliseconds

•••••

E.12.4 Statistics

•••••

Jitter

StatisticID: jit (0x0007)

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

Type: double

Possible values: any 64-bit integer

Delay

StatisticID:delay (0x0008)

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

Type: double

Possible values: any 64-bit integer

•••••

E.13.1 Properties

•••••

Gain Control

PropertyID: gain (0x000a)

Gain control, or usage of signal level adaptation and noise level reduction is used to adapt the level of the outbound signal. However, it is necessary, for example for modem calls, to turn off this function. When the value is set to "automatic", the termination serves as an automatic level control (ALC) with a target level provisioned on the MG and the direction being outward.

Type: integer

Possible values:

The gain control specifies the gain in decibels (positive or negative), with the maximum positive integer, 214748647 (0x7fffffff), reserved to represent "automatic" ~~parameter may either be specified as "automatic" (0xffffffff), or as an explicit number of decibels of gain (any other integer value)~~. The default value is provisioned in the MG.

Defined in: LocalControlDescriptor

Characteristics: read/write

•••••

11) Appendix I Example call flows

•••••

Change Appendix I as follows:

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, Version=2,
            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. Local and Remote are assumed to be provisioned.

```
MGC to MG1:
MEGACO/~~1~~2 [123.123.123.4]:55555
Transaction = 9999 {
    Context = - {
        Modify = A4444 {
            Media { Stream = 1 {
                     LocalControl {
                         Mode = SendReceive,
                         tdmc/gain=2,  ; in dB,
                         tdmc/ec=on
                     },
                 }
            },
            Events = 2222 {al/of ~~(~~{strict=state}~~)~~}
        }
    }
}
```

The dialplan script could have been loaded into the MG previously. Its function would be to wait for the 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~~2 [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~~2 [124.124.124.222]:55555
Transaction = 10000 {
   Context = - {
       Notify = A4444 {ObservedEvents =2222 {
         19990729T22000000:al/of(init=false)}}
   }
}
```

7) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/~~1~~2 [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~~2 [123.123.123.4]:55555
Transaction = 10001 {
    Context = - {
        Modify = A4444 {
            Events = 2223 {
                al/on(strict=state), dd/ce {DigitMap=Dialplan0}
            },
            Signals {cg/dt},
            DigitMap= Dialplan0{
(0| 00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxxx|9011x.)}
        }
    }
}
```

9) And the Modify is acknowledged.

```
MG1 to MGC:
MEGACO/~~1~~2 [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~~2 [124.124.124.222]:55555
Transaction = 10002 {
   Context = - {
       Notify = A4444 {ObservedEvents =2223 {
         19990729T22010001:dd/ce{ds="916135551212",Meth=UM}}}
   }
}
```

11) And the Notify is acknowledged.

```
MGC to MG1:
MEGACO/~~1~~2 [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~~2 [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~~2 [124.124.124.222]:55555
Reply = 10003 {
   Context = 2000 {
      Add = A4444,
      Add=A4445{
         Media {
             Stream = 1 {
                 Local {
v=0
o=- 2890844526 2890842807 IN IP4 124.124.124.222
s=-
t= 0 0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP 4
a=ptime:30
a=recvonly
                 } ; RTP profile for G.723.1 is 4
             }
         }
      }
   }
}
```

14) The MGC will now associate A5555 with a new Context on MG2, and establish an RTP Stream (i.e. A5556 will be assigned), SendReceive connection through to the originating user, User 1. The MGC also sets ring on A5555.

```
MGC to MG2:
MEGACO/~~1~~2 [123.123.123.4]:55555
Transaction = 50003 {
    Context = $ {
       Add = A5555  { Media {
            Stream = 1 {
                 LocalControl {Mode = SendReceive} }},
          Events=1234{al/of(strict=state)},
            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~~2 [125.125.125.111]:55555
Reply = 50003 {
   Context = 5000 {
     Add = A5555,
      Add = A5556{
         Media {
            Stream = 1 {
                Local {
v=0

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP 4
}
            } ; RTP profile for G.723.1 is 4
         }
       }
   }
}
```

16) The above IPAddr and UDPport need to be given to MG1 now.

```
MGC to MG1:
MEGACO/~~1~~2 [123.123.123.4]:55555
Transaction = 10005 {
  Context = 2000 {
    Modify = A4444 {
      Signals {cg/rt}
    },
    Modify = A4445 {
       Media {
            Stream = 1 {
                Remote {
v=0

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP 4
                }
            } ; RTP profile for G.723.1 is 4
        }
    }
  }
}

MG1 to MGC:
MEGACO/~~1~~2 [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~~2 [125.125.125.111]:55555
Transaction = 50005 {
   Context = 5000 {
       Notify = A5555 {ObservedEvents =1234 {
         19990729T22020002:al/of(init=false)}}
   }
}

From MGC to MG2:

MEGACO/~~1~~2 [123.123.123.4]:55555
Reply = 50005 {
    Context = - {Notify = A5555}
}

From MGC to MG2:

MEGACO/~~1~~2 [123.123.123.4]:55555
Transaction = 50006 {
   Context = 5000 {
      Modify = A5555 {
         Events = 1235 {al/on(strict=state)},
          Signals { } ; to turn off ringing
      }
   }
}

From MG2 to MGC:

MEGACO/~~1~~2 [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~~2 [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~~2 [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~~2 [123.123.123.4]:55555
Transaction = 50007 {
   Context = - {AuditValue = A5556{
      Audit{Media, DigitMap, Events, Signals, Packages, Statistics }}
   }
}
```

20) The MG2 replies.

```
MEGACO/~~1~~2 [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

o=- 7736844526 7736842807 IN IP4 125.125.125.111
s=-
t= 0 0
c=IN IP4 125.125.125.111
m=audio 1111 RTP/AVP  4
a=ptime:30
                },
                 Remote {
v=0

o=- 2890844526 2890842807 IN IP4 124.124.124.222
s=-
t= 0 0
c=IN IP4 124.124.124.222
m=audio 2222 RTP/AVP  4
a=ptime:30
                 } } },
          Events,
          Signals,
          DigitMap,
          Packages {nt-1, rtp-1},
          Statistics { rtp/ps=1200,  ; packets sent
                       nt/os=62300, ; octets sent
                       rtp/pr=700, ; packets received
                       nt/or=45100, ; octets received
                       rtp/pl=0.2,  ; % packet loss
                       rtp/jit=20,
                       rtp/delay=40 } ; avg latency
       }
    }
}
```

21) When the MGC receives an 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~~2 [125.125.125.111]:55555
Transaction = 50008 {
   Context = 5000 {
       Notify = A5555 {ObservedEvents =1235 {
          19990729T24020002:al/on(init=false)}
       }
   }
}

From MGC to MG2:

MEGACO/~~1~~2 [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~~2 [123.123.123.4]:55555
Transaction = 50009 {
   Context = 5000 {
      Subtract = A5555 {Audit{Statistics}},
      Subtract = A5556 {Audit{Statistics}}
   }
}
```

```
From MG2 to MGC:

MEGACO/~~1~~ 2 [125.125.125.111]:55555
Reply = 50009 {
   Context = 5000 {
     Subtract = A5555 {
          Statistics {
             nt/os=45123, ; Octets Sent
             nt/or=45123, ; Octets Received
             nt/dur=40000 ; in milliseconds
             }
       },
       Subtract = A5556 {
          Statistics {
             rtp/ps=1245,  ; packets sent
             nt/os=62345,  ; octets sent
             rtp/pr=780,   ; packets received
             nt/or=45123,  ; octets received
             rtp/pl=10,    ; ~~%~~ packets lost
             rtp/jit=27,
             rtp/delay=48, ; average latency
             nt/dur=38000  ; in millisec
          }
       }
   }
}
```

23) The MGC now sets up both MG1 and MG2 to be ready to detect the next off-hook event. See step 1). Note that this could be the default state of a termination in the null context, and if this were the case, no message need be sent from the MGC to the MG. Once a termination returns to the null context, it goes back to the default termination values for that termination.

Printed in Switzerland
Geneva, 2004