Network Working Group
Request for Comments: 3435
Obsoletes: 2705
Category: Informational

F. Andreasen
B. Foster
Cisco Systems

January 2003

Media Gateway Control Protocol (MGCP) Version 1.0

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

This document is being published for the information of the community. It describes a protocol that is currently being deployed in a number of products. Implementers should be aware of RFC 3015, which was developed in the IETF Megaco Working Group and the ITU-T SG16 and which is considered by the IETF and ITU-T to be the standards-based (including reviewed security considerations) way to meet the needs that MGCP was designed to address.

Abstract

This document describes an application programming interface and a corresponding protocol (MGCP) which is used between elements of a decomposed multimedia gateway. The decomposed multimedia gateway consists of a Call Agent, which contains the call control "intelligence", and a media gateway which contains the media functions, e.g., conversion from TDM voice to Voice over IP.

Media gateways contain endpoints on which the Call Agent can create, modify and delete connections in order to establish and control media sessions with other multimedia endpoints. Also, the Call Agent can instruct the endpoints to detect certain events and generate signals. The endpoints automatically communicate changes in service state to the Call Agent. Furthermore, the Call Agent can audit endpoints as well as the connections on endpoints.

The basic and general MGCP protocol is defined in this document, however most media gateways will need to implement one or more MGCP packages, which define extensions to the protocol suitable for use with specific types of media gateways. Such packages are defined in separate documents.

Table of Contents

1. Introduction

This document describes an abstract application programming interface (MGCI) and a corresponding protocol (MGCP) for controlling media gateways from external call control elements called media gateway controllers or Call Agents. A media gateway is typically a network element that provides conversion between the audio signals carried on telephone circuits and data packets carried over the Internet or over other packet networks. Examples of media gateways are:

* Trunking gateways, that interface between the telephone network and a Voice over IP network. Such gateways typically manage a large number of digital circuits.

* Voice over ATM gateways, which operate much the same way as voice over IP trunking gateways, except that they interface to an ATM network.

* Residential gateways, that provide a traditional analog (RJ11) interface to a Voice over IP network. Examples of residential gateways include cable modem/cable set-top boxes, xDSL devices, and broad-band wireless devices.

* Access gateways, that provide a traditional analog (RJ11) or digital PBX interface to a Voice over IP network. Examples of access gateways include small-scale voice over IP gateways.

* Business gateways, that provide a traditional digital PBX interface or an integrated "soft PBX" interface to a Voice over IP network.

* Network Access Servers, that can attach a "modem" to a telephone circuit and provide data access to the Internet. We expect that in the future, the same gateways will combine Voice over IP services and Network Access services.

* Circuit switches, or packet switches, which can offer a control interface to an external call control element.

MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements known as Call Agents. The MGCP assumes that these call control elements, or Call Agents, will synchronize with each other to send coherent commands and responses to the gateways under their control. If this assumption is violated, inconsistent behavior should be expected. MGCP does not define a mechanism for synchronizing Call Agents. MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents. In consequence, this document specifies in great

detail the expected behavior of the gateways, but only specifies those parts of a Call Agent implementation, such as timer management, that are mandated for proper operation of the protocol.

MGCP assumes a connection model where the basic constructs are endpoints and connections. Endpoints are sources and/or sinks of data and can be physical or virtual. Examples of physical endpoints are:

* An interface on a gateway that terminates a trunk connected to a PSTN switch (e.g., Class 5, Class 4, etc.). A gateway that terminates trunks is called a trunking gateway.

* An interface on a gateway that terminates an analog POTS connection to a phone, key system, PBX, etc. A gateway that terminates residential POTS lines (to phones) is called a residential gateway.

An example of a virtual endpoint is an audio source in an audio-content server. Creation of physical endpoints requires hardware installation, while creation of virtual endpoints can be done by software.

Connections may be either point to point or multipoint. A point to point connection is an association between two endpoints with the purpose of transmitting data between these endpoints. Once this association is established for both endpoints, data transfer between these endpoints can take place. A multipoint connection is established by connecting the endpoint to a multipoint session.

Connections can be established over several types of bearer networks, for example:

* Transmission of audio packets using RTP and UDP over an IP network.

* Transmission of audio packets using AAL2, or another adaptation layer, over an ATM network.

* Transmission of packets over an internal connection, for example the TDM backplane or the interconnection bus of a gateway. This is used, in particular, for "hairpin" connections, connections that terminate in a gateway but are immediately rerouted over the telephone network.

For point-to-point connections the endpoints of a connection could be in separate gateways or in the same gateway.

1.1 Relation with the H.323 Standards

MGCP is designed as an internal protocol within a distributed system that appears to the outside as a single VoIP gateway. This system is composed of a Call Agent, that may or may not be distributed over several computer platforms, and of a set of gateways, including at least one "media gateway" that perform the conversion of media signals between circuits and packets, and at least one "signaling gateway" when connecting to an SS7 controlled network. In a typical configuration, this distributed gateway system will interface on one side with one or more telephony (i.e., circuit) switches, and on the other side with H.323 conformant systems, as indicated in the following table:

Functional Plane	Phone switch	Terminating Entity	H.323 conformant systems
Signaling Plane	Signaling exchanges through SS7/ISUP	Call agent	Signaling exchanges with the Call Agent through H.225/RAS and H.225/Q.931.
			Possible negotiation of logical channels and transmission parameters through H.245 with the call agent.
		Internal synchronization through MGCP	
Bearer Data Transport Plane	Connection through high speed trunk groups	Telephony gateways	Transmission of VoIP data using RTP directly between the H.323 station and the gateway.

In the MGCP model, the gateways focus on the audio signal translation function, while the Call Agent handles the call signaling and call processing functions. As a consequence, the Call Agent implements the "signaling" layers of the H.323 standard, and presents itself as an "H.323 Gatekeeper" or as one or more "H.323 Endpoints" to the H.323 systems.

1.2 Relation with the IETF Standards

While H.323 is the recognized standard for VoIP terminals, the IETF has also produced specifications for other types of multi-media applications. These other specifications include:

* the Session Description Protocol (SDP), RFC 2327

* the Session Announcement Protocol (SAP), RFC 2974

* the Session Initiation Protocol (SIP), RFC 3261

* the Real Time Streaming Protocol (RTSP), RFC 2326.

The latter three specifications are in fact alternative signaling standards that allow for the transmission of a session description to an interested party. SAP is used by multicast session managers to distribute a multicast session description to a large group of recipients, SIP is used to invite an individual user to take part in a point-to-point or unicast session, RTSP is used to interface a server that provides real time data. In all three cases, the session description is described according to SDP; when audio is transmitted, it is transmitted through the Real-time Transport Protocol, RTP.

The distributed gateway systems and MGCP will enable PSTN telephony users to access sessions set up using SAP, SIP or RTSP. The Call Agent provides for signaling conversion, according to the following table:

Functional Plane	Phone switch	Terminating Entity	IETF conforming systems
Signaling Plane	Signaling exchanges through SS7/ISUP	Call agent	Signaling exchanges with the Call Agent through SAP, SIP or RTSP.
			Negotiation of session description parameters through SDP (telephony gateway terminated but passed via the call agent to and from the IETF conforming system)
		Internal synchronization through MGCP	
Bearer Data Transport Plane	Connection through high speed trunk groups	Telephony gateways	Transmission of VoIP data using RTP, directly between the remote IP end system and the gateway.

The SDP standard has a pivotal status in this architecture. We will see in the following description that we also use it to carry session descriptions in MGCP.

1.3 Definitions

Trunk: A communication channel between two switching systems, e.g., a DS0 on a T1 or E1 line.

1.4 Conventions used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [2].

2. Media Gateway Control Interface

The interface functions provide for connection control and endpoint control. Both use the same system model and the same naming conventions.

2.1 Model and Naming Conventions

The MGCP assumes a connection model where the basic constructs are endpoints and connections. Connections are grouped in calls. One or more connections can belong to one call. Connections and calls are set up at the initiative of one or more Call Agents.

2.1.1 Types of Endpoints

In the introduction, we presented several classes of gateways. Such classifications, however, can be misleading. Manufacturers can arbitrarily decide to provide several types of services in a single package. A single product could well, for example, provide some trunk connections to telephony switches, some primary rate connections and some analog line interfaces, thus sharing the characteristics of what we described in the introduction as "trunking", "access" and "residential" gateways. MGCP does not make assumptions about such groupings. We simply assume that media gateways support collections of endpoints. The type of the endpoint determines its functionality. Our analysis, so far, has led us to isolate the following basic endpoint types:

* Digital channel (DS0),

* Analog line,

* Announcement server access point,

* Interactive Voice Response access point,

* Conference bridge access point,

* Packet relay,

* ATM "trunk side" interface.

In this section, we will describe the expected behavior of such endpoints.

This list is not final. There may be other types of endpoints defined in the future, for example test endpoints that could be used to check network quality, or frame-relay endpoints that could be used to manage audio channels multiplexed over a frame-relay virtual circuit.

2.1.1.1 Digital Channel (DS0)

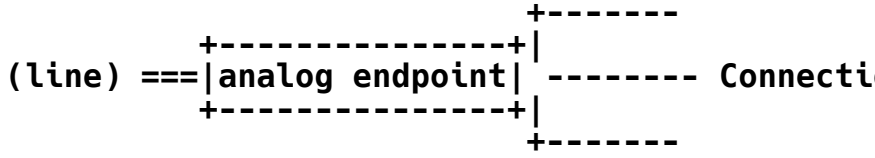
Digital channels provide a 64 Kbps service. Such channels are found in trunk and ISDN interfaces. They are typically part of digital multiplexes, such as T1, E1, T3 or E3 interfaces. Media gateways that support such channels are capable of translating the digital signals received on the channel, which may be encoded according to A-law or mu-law, using either the complete set of 8 bits per sample or only 7 of these bits, into audio packets. When the media gateway also supports a Network Access Server (NAS) service, the gateway shall be capable of receiving either audio-encoded data (modem connection) or binary data (ISDN connection) and convert them into data packets.

Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint supports is a characteristic of the gateway, and may in fact vary according to the allocation of resources within the gateway.

In some cases, digital channels are used to carry signaling. This is the case for example for SS7 "F" links, or ISDN "D" channels. Media gateways that support these signaling functions shall be able to send and receive the signaling packets to and from a Call Agent, using the "backhaul" procedures defined by the SIGTRAN working group of the IETF. Digital channels are sometimes used in conjunction with channel associated signaling, such as "MF R2". Media gateways that support these signaling functions shall be able to detect and produce the corresponding signals, such as for example "wink" or "A", according to the event signaling and reporting procedures defined in MGCP.

2.1.1.2 Analog Line

Analog lines can be used either as a "client" interface, providing service to a classic telephone unit, or as a "service" interface, allowing the gateway to send and receive analog calls. When the media gateway also supports a NAS service, the gateway shall be capable of receiving audio-encoded data (modem connection) and convert them into data packets.



Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The audio signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint supports is a characteristic of the gateway, and may in fact vary according to the allocation of resources within the gateway. A typical gateway should however be able to support two or three connections per endpoint, in order to support services such as "call waiting" or "three way calling".

2.1.1.3 Announcement Server Access Point

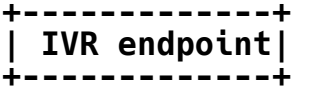
An announcement server endpoint provides access to an announcement service. Under requests from the Call Agent, the announcement server will "play" a specified announcement. The requests from the Call Agent will follow the event signaling and reporting procedures defined in MGCP.

A given announcement endpoint is not expected to support more than one connection at a time. If several connections were established to the same endpoint, then the same announcements would be played simultaneously over all the connections.

Connections to an announcement server are typically one way, or "half duplex" -- the announcement server is not expected to listen to the audio signals from the connection.

2.1.1.4 Interactive Voice Response Access Point

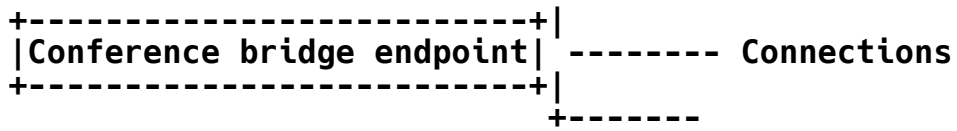
An Interactive Voice Response (IVR) endpoint provides access to an IVR service. Under requests from the Call Agent, the IVR server will "play" announcements and tones, and will "listen" to responses, such as DTMF input or voice messages, from the user. The requests from the Call Agent will follow the event signaling and reporting procedures defined in MGCP.



A given IVR endpoint is not expected to support more than one connection at a time. If several connections were established to the same endpoint, then the same tones and announcements would be played simultaneously over all the connections.

2.1.1.5 Conference Bridge Access Point

A conference bridge endpoint is used to provide access to a specific conference.



Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint supports is a characteristic of the gateway, and may in fact vary according to the allocation of resources within the gateway.

2.1.1.6 Packet Relay

A packet relay endpoint is a specific form of conference bridge, that typically only supports two connections. Packets relays can be found in firewalls between a protected and an open network, or in transcoding servers used to provide interoperation between incompatible gateways, for example gateways that do not support compatible compression algorithms, or gateways that operate over different transmission networks such as IP and ATM.

2.1.1.7 ATM "trunk side" Interface

ATM "trunk side" endpoints are typically found when one or several ATM permanent virtual circuits are used as a replacement for the classic "TDM" trunks linking switches. When ATM/AAL2 is used, several trunks or channels are multiplexed on a single virtual circuit; each of these trunks correspond to a single endpoint.

Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint supports is a characteristic of the gateway, and may in fact vary according to the allocation of resources within the gateway.

2.1.2 Endpoint Identifiers

Endpoint identifiers have two components that both are case-insensitive:

* the domain name of the gateway that is managing the endpoint

* a local name within that gateway

Endpoint names are of the form:

   local-endpoint-name@domain-name

where domain-name is an absolute domain-name as defined in RFC 1034 and includes a host portion, thus an example domain-name could be:

   mygateway.whatever.net

Also, domain-name may be an IP-address of the form defined for domain name in RFC 821, thus another example could be (see RFC 821 for details):

[192.168.1.2]

Both IPv4 and IPv6 addresses can be specified, however use of IP addresses as endpoint identifiers is generally discouraged.

Note that since the domain name portion is part of the endpoint identifier, different forms or different values referring to the same entity are not freely interchangeable. The most recently supplied form and value MUST always be used.

The local endpoint name is case-insensitive. The syntax of the local endpoint name is hierarchical, where the least specific component of the name is the leftmost term, and the most specific component is the rightmost term. The precise syntax depends on the type of endpoint being named and MAY start with a term that identifies the endpoint type. In any case, the local endpoint name MUST adhere to the following naming rules:

1) The individual terms of the naming path MUST be separated by a single slash ("/", ASCII 2F hex).

2) The individual terms are character strings composed of letters, digits or other printable characters, with the exception of characters used as delimiters ("/", "@"), characters used for wildcarding ("*", "$") and white spaces.

3) Wild-carding is represented either by an asterisk ("*") or a dollar sign ("$") for the terms of the naming path which are to be wild-carded. Thus, if the full local endpoint name is of the form:

    term1/term2/term3

   then the entity name field looks like this depending on which terms are wild-carded:

    */term2/term3 if term1 is wild-carded
    term1/*/term3 if term2 is wild-carded
    term1/term2/* if term3 is wild-carded
    term1/*/*   if term2 and term3 are wild-carded, etc.

   In each of these examples a dollar sign could have appeared instead of an asterisk.

4) A term represented by an asterisk ("*") is to be interpreted as: "use ALL values of this term known within the scope of the Media Gateway". Unless specified otherwise, this refers to all endpoints configured for service, regardless of their actual service state, i.e., in-service or out-of-service.

5) A term represented by a dollar sign ("$") is to be interpreted as: "use ANY ONE value of this term known within the scope of the Media Gateway". Unless specified otherwise, this only refers to endpoints that are in-service.

Furthermore, it is RECOMMENDED that Call Agents adhere to the following:

* Wild-carding should only be done from the right, thus if a term is wild-carded, then all terms to the right of that term should be wild-carded as well.

* In cases where mixed dollar sign and asterisk wild-cards are used, dollar-signs should only be used from the right, thus if a term had a dollar sign wild-card, all terms to the right of that term should also contain dollar sign wild-cards.

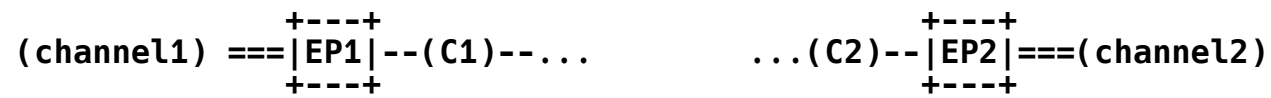
The description of a specific command may add further criteria for selection within the general rules given above.

Note, that wild-cards may be applied to more than one term in which case they shall be evaluated from left to right. For example, if we have the endpoint names "a/1", "a/2", "b/1", and "b/2", then "$/*" (which is not recommended) will evaluate to either "a/1, a/2", or "b/1, b/2". However, "*/$" may evaluate to "a/1, b/1", "a/1, b/2", "a/2, b/1", or "a/2, b/2". The use of mixed wild-cards in a command is considered error prone and is consequently discouraged.

A local name that is composed of only a wildcard character refers to either all (*) or any ($) endpoints within the media gateway.

2.1.3 Calls and Connections

Connections are created on the Call Agent on each endpoint that will be involved in the "call". In the classic example of a connection between two "DS0" endpoints (EP1 and EP2), the Call Agents controlling the endpoints will establish two connections (C1 and C2):



Each connection will be designated locally by an endpoint unique connection identifier, and will be characterized by connection attributes.

When the two endpoints are located on gateways that are managed by the same Call Agent, the creation is done via the three following steps:

1) The Call Agent asks the first gateway to "create a connection" on the first endpoint. The gateway allocates resources to that connection, and responds to the command by providing a "session description". The session description contains the information necessary for a third party to send packets towards the newly created connection, such as for example IP address, UDP port, and codec parameters.

2) The Call Agent then asks the second gateway to "create a connection" on the second endpoint. The command carries the "session description" provided by the first gateway. The gateway allocates resources to that connection, and responds to the command by providing its own "session description".

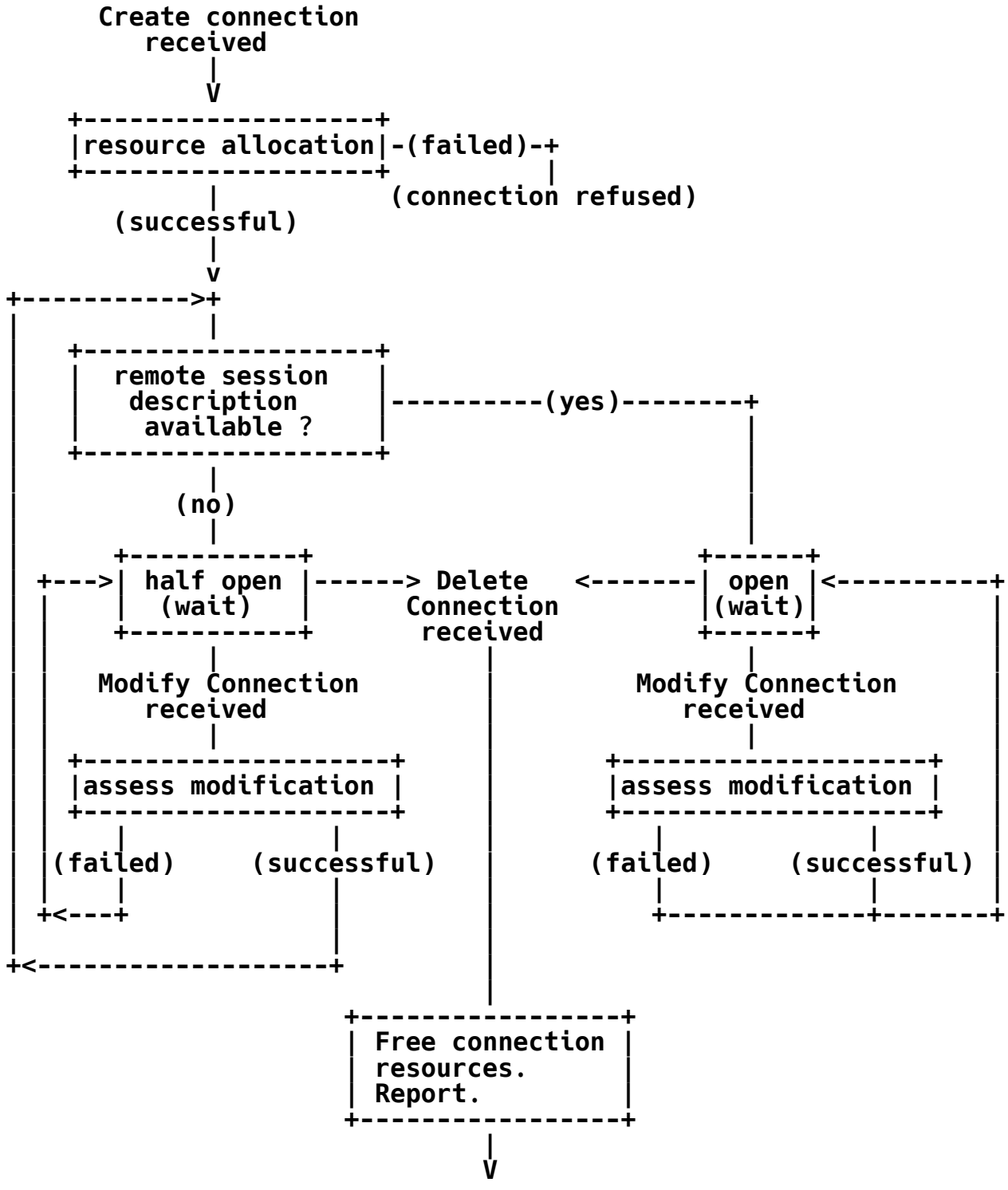
3) The Call Agent then uses a "modify connection" command to provide this second "session description" to the first endpoint. Once this is done, communication can proceed in both directions.

When the two endpoints are located on gateways that are managed by two different Call Agents, the Call Agents exchange information through a Call-Agent to Call-Agent signaling protocol, e.g., SIP [7], in order to synchronize the creation of the connection on the two endpoints.

Once a connection has been established, the connection parameters can be modified at any time by a "modify connection" command. The Call Agent may for example instruct the gateway to change the codec used on a connection, or to modify the IP address and UDP port to which data should be sent, if a connection is "redirected".

The Call Agent removes a connection by sending a "delete connection" command to the gateway. The gateway may also, under some circumstances, inform a gateway that a connection could not be sustained.

The following diagram provides a view of the states of a connection, as seen from the gateway:


Create connection received
resource allocation
(failed)
(connection refused)
(successful)
remote session description available ?
(yes)
(no)
half open (wait)
Delete Connection received
open (wait)
Modify Connection received
assess modification
(failed)
(successful)
Modify Connection received
assess modification
(failed)
(successful)
Free connection resources. Report.

2.1.3.1 Names of Calls

One of the attributes of each connection is the "call identifier", which as far as the MGCP protocol is concerned has little semantic meaning, and is mainly retained for backwards compatibility.

Calls are identified by unique identifiers, independent of the underlying platforms or agents. Call identifiers are hexadecimal strings, which are created by the Call Agent. The maximum length of call identifiers is 32 characters.

Call identifiers are expected to be unique within the system, or at a minimum, unique within the collection of Call Agents that control the same gateways. From the gateway's perspective, the Call identifier is thus unique. When a Call Agent builds several connections that pertain to the same call, either on the same gateway or in different gateways, these connections that belong to the same call should share the same call-id. This identifier can then be used by accounting or management procedures, which are outside the scope of MGCP.

2.1.3.2 Names of Connections

Connection identifiers are created by the gateway when it is requested to create a connection. They identify the connection within the context of an endpoint. Connection identifiers are treated in MGCP as hexadecimal strings. The gateway MUST make sure that a proper waiting period, at least 3 minutes, elapses between the end of a connection that used this identifier and its use in a new connection for the same endpoint (gateways MAY decide to use identifiers that are unique within the context of the gateway). The maximum length of a connection identifier is 32 characters.

2.1.3.3 Management of Resources, Attributes of Connections

Many types of resources will be associated to a connection, such as specific signal processing functions or packetization functions. Generally, these resources fall in two categories:

1) Externally visible resources, that affect the format of "the bits on the network" and must be communicated to the second endpoint involved in the connection.

2) Internal resources, that determine which signal is being sent over the connection and how the received signals are processed by the endpoint.

The resources allocated to a connection, and more generally the handling of the connection, are chosen by the gateway under instructions from the Call Agent. The Call Agent will provide these instructions by sending two sets of parameters to the gateway:

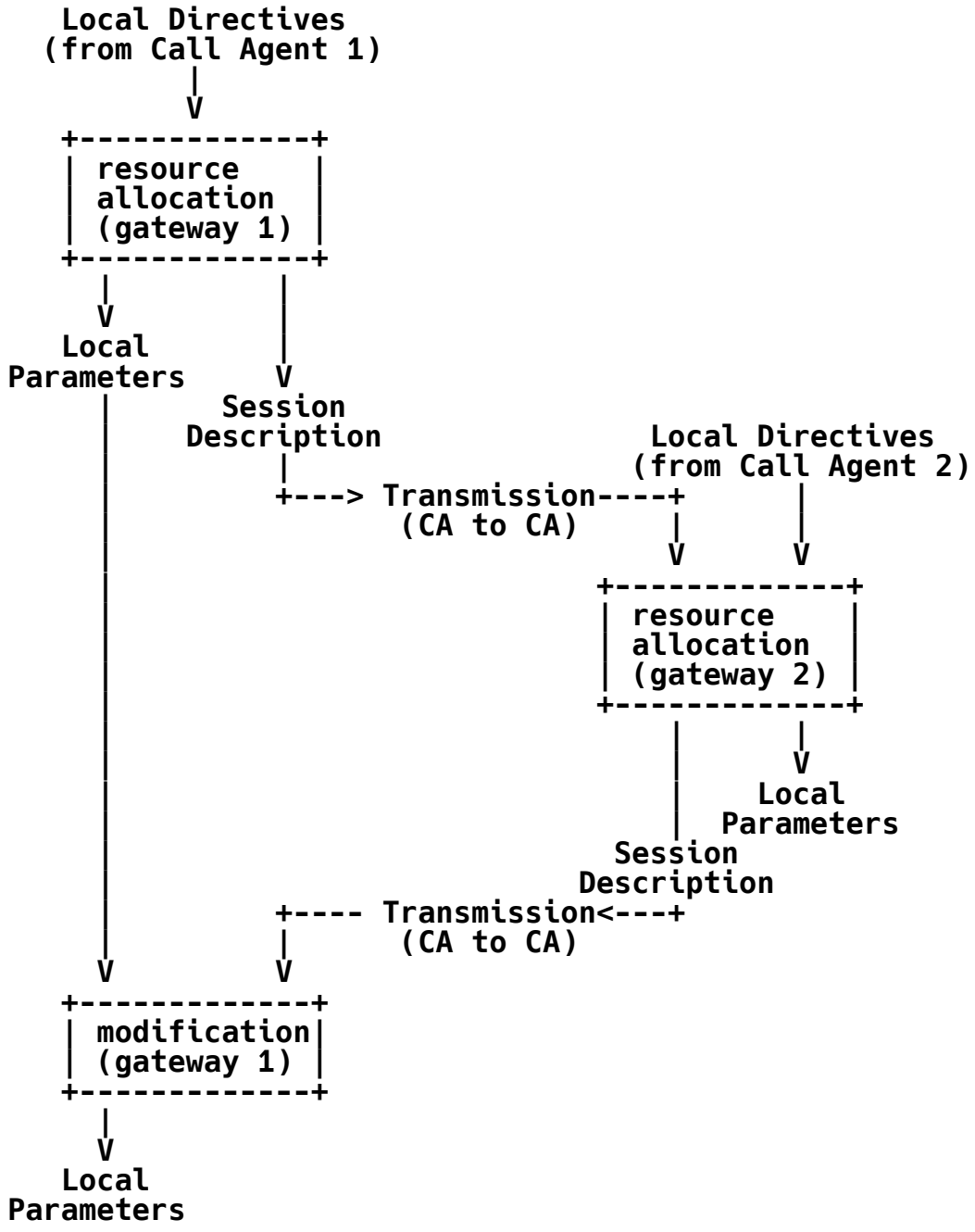
1) The local directives instruct the gateway on the choice of resources that should be used for a connection,

2) When available, the "session description" provided by the other end of the connection (referred to as the remote session description).

The local directives specify such parameters as the mode of the connection (e.g., send-only, or send-receive), preferred coding or packetization methods, usage of echo cancellation or silence suppression. (A detailed list can be found in the specification of the LocalConnectionOptions parameter of the CreateConnection command.) Depending on the parameter, the Call Agent MAY either specify a value, a range of values, or no value at all. This allows various implementations to implement various levels of control, from a very tight control where the Call Agent specifies minute details of the connection handling to a very loose control where the Call Agent only specifies broad guidelines, such as the maximum bandwidth, and lets the gateway choose the detailed values subject to the guidelines.

Based on the value of the local directives, the gateway will determine the resources to allocate to the connection. When this is possible, the gateway will choose values that are in line with the remote session description - but there is no absolute requirement that the parameters be exactly the same.

Once the resources have been allocated, the gateway will compose a "session description" that describes the way it intends to send and receive packets. Note that the session description may in some cases present a range of values. For example, if the gateway is ready to accept one of several compression algorithms, it can provide a list of these accepted algorithms.



-- Information flow: local directives & session descriptions --

2.1.3.4 Special Case of Local Connections

Large gateways include a large number of endpoints which are often of different types. In some networks, we may often have to set-up connections between endpoints that are located within the same gateway. Examples of such connections may be:

* Connecting a call to an Interactive Voice-Response unit,

* Connecting a call to a Conferencing unit,

* Routing a call from one endpoint to another, something often described as a "hairpin" connection.

Local connections are much simpler to establish than network connections. In most cases, the connection will be established through some local interconnecting device, such as for example a TDM bus.

When two endpoints are managed by the same gateway, it is possible to specify the connection in a single command that conveys the names of the two endpoints that will be connected. The command is essentially a "Create Connection" command which includes the name of the second endpoint in lieu of the "remote session description".

2.1.4 Names of Call Agents and Other Entities

The media gateway control protocol has been designed to allow the implementation of redundant Call Agents, for enhanced network reliability. This means that there is no fixed binding between entities and hardware platforms or network interfaces.

Call Agent names consist of two parts, similar to endpoint names. Semantically, the local portion of the name does not exhibit any internal structure. An example Call Agent name is:

```
ca1@ca.whatever.net
```

Note that both the local part and the domain name have to be supplied. Nevertheless, implementations are encouraged to accept call agent names consisting of only the domain name.

Reliability can be improved by using the following procedures:

* Entities such as endpoints or Call Agents are identified by their domain name, not their network addresses. Several addresses can be

associated with a domain name. If a command or a response cannot be forwarded to one of the network addresses, implementations MUST retry the transmission using another address.

* Entities MAY move to another platform. The association between a logical name (domain name) and the actual platform is kept in the domain name service. Call Agents and Gateways MUST keep track of the time-to-live of the record they read from the DNS. They MUST query the DNS to refresh the information if the time to live has expired.

In addition to the indirection provided by the use of domain names and the DNS, the concept of "notified entity" is central to reliability and fail-over in MGCP. The "notified entity" for an endpoint is the Call Agent currently controlling that endpoint. At any point in time, an endpoint has one, and only one, "notified entity" associated with it. The "notified entity" determines where the endpoint will send commands to; when the endpoint needs to send a command to the Call Agent, it MUST send the command to its current "notified entity". The "notified entity" however does not determine where commands can be received from; any Call Agent can send commands to the endpoint. Please refer to Section 5 for the relevant security considerations.

Upon startup, the "notified entity" MUST be set to a provisioned value. Most commands sent by the Call Agent include the ability to explicitly name the "notified entity" through the use of a "NotifiedEntity" parameter. The "notified entity" will stay the same until either a new "NotifiedEntity" parameter is received or the endpoint does a warm or cold (power-cycle) restart.

If a "NotifiedEntity" parameter is sent with an "empty" value, the "notified entity" for the endpoint will be set to empty. If the "notified entity" for an endpoint is empty or has not been set explicitly (neither by a command nor by provisioning), the "notified entity" will then default to the source address (i.e., IP address and UDP port number) of the last successful non-audit command received for the endpoint. Auditing will thus not change the "notified entity". Use of an empty "NotifiedEntity" parameter value is strongly discouraged as it is error prone and eliminates the DNS-based fail-over and reliability mechanisms.

2.1.5 Digit Maps

The Call Agent can ask the gateway to collect digits dialed by the user. This facility is intended to be used with residential gateways to collect the numbers that a user dials; it can also be used with

trunking gateways and access gateways alike, to collect access codes, credit card numbers and other numbers requested by call control services.

One procedure is for the gateway to notify the Call Agent of each individual dialed digit, as soon as they are dialed. However, such a procedure generates a large number of interactions. It is preferable to accumulate the dialed numbers in a buffer, and to transmit them in a single message.

The problem with this accumulation approach, however, is that it is hard for the gateway to predict how many numbers it needs to accumulate before transmission. For example, using the phone on our desk, we can dial the following numbers:

0	Local operator
00	Long distance operator
xxxx	Local extension number
8xxxxxxx	Local number
#xxxxxxx	Shortcut to local number at other corporate sites
*xx	Star services
91xxxxxxxxxx	Long distance number
9011 + up to 15 digits	International number

The solution to this problem is to have the Call Agent load the gateway with a digit map that may correspond to the dial plan. This digit map is expressed using a syntax derived from the Unix system command, egrep. For example, the dial plan described above results in the following digit map:

```
(0T|00T|[1-7]xxx|8xxxxxxx|#xxxxxxx|*xx|91xxxxxxxxxx|9011x.T)
```

The formal syntax of the digit map is described by the DigitMap rule in the formal syntax description of the protocol (see Appendix A) - support for basic digit map letters is REQUIRED while support for extension digit map letters is OPTIONAL. A gateway receiving a digit map with an extension digit map letter not supported SHOULD return error code 537 (unknown digit map extension).

A digit map, according to this syntax, is defined either by a (case insensitive) "string" or by a list of strings. Each string in the list is an alternative numbering scheme, specified either as a set of digits or timers, or as an expression over which the gateway will attempt to find a shortest possible match. The following constructs can be used in each numbering scheme:

* Digit: A digit from "0" to "9".
* Timer: The symbol "T" matching a timer expiry.
* DTMF: A digit, a timer, or one of the symbols "A", "B", "C", "D", "#", or "*". Extensions may be defined.
* Wildcard: The symbol "x" which matches any digit ("0" to "9").
* Range: One or more DTMF symbols enclosed between square brackets ("[" and "]").
* Subrange: Two digits separated by hyphen ("-") which matches any digit between and including the two. The subrange construct can only be used inside a range construct, i.e., between "[" and "]".
* Position: A period (".") which matches an arbitrary number, including zero, of occurrences of the preceding construct.

A gateway that detects events to be matched against a digit map MUST do the following:

1) Add the event code as a token to the end of an internal state variable for the endpoint called the "current dial string".

2) Apply the current dial string to the digit map table, attempting a match to each expression in the digit map.

3) If the result is under-qualified (partially matches at least one entry in the digit map and doesn't completely match another entry), do nothing further.

If the result matches an entry, or is over-qualified (i.e., no further digits could possibly produce a match), send the list of accumulated events to the Call Agent. A match, in this specification, can be either a "perfect match," exactly matching one of the specified alternatives, or an impossible match, which occurs when the dial string does not match any of the alternatives. Unexpected timers, for example, can cause "impossible matches". Both perfect matches and impossible matches trigger notification of the accumulated digits (which may include other events - see Section 2.3.3).

The following example illustrates the above. Assume we have the digit map:

```
(xxxxxxx|x11)
```

and a current dial string of "41". Given the input "1" the current dial string becomes "411". We have a partial match with "xxxxxxx", but a complete match with "x11", and hence we send "411" to the Call Agent.

The following digit map example is more subtle:

```
(0[12].|00|1[12].1|2x.#)
```

Given the input "0", a match will occur immediately since position (".") allows for zero occurrences of the preceding construct. The input "00" can thus never be produced in this digit map.

Given the input "1", only a partial match exists. The input "12" is also only a partial match, however both "11" and "121" are a match.

Given the input "2", a partial match exists. A partial match also exists for the input "23", "234", "2345", etc. A full match does not occur here until a "#" is generated, e.g., "2345#". The input "2#" would also have been a match.

Note that digit maps simply define a way of matching sequences of event codes against a grammar. Although digit maps as defined here are for DTMF input, extension packages can also be defined so that digit maps can be used for other types of input represented by event codes that adhere to the digit map syntax already defined for these event codes (e.g., "1" or "T"). Where such usage is envisioned, the definition of the particular event(s) SHOULD explicitly state that in the package definition.

Since digit maps are not bounded in size, it is RECOMMENDED that gateways support digit maps up to at least 2048 bytes per endpoint.

2.1.6 Packages

MGCP is a modular and extensible protocol, however with extensibility comes the need to manage, identify, and name the individual extensions. This is achieved by the concept of packages, which are simply well-defined groupings of extensions. For example, one package may support a certain group of events and signals, e.g., off-hook and ringing, for analog access lines. Another package may support another group of events and signals for analog access lines or for another type of endpoint such as video. One or more packages may be supported by a given endpoint.

MGCP allows the following types of extensions to be defined in a package:

* BearerInformation

* LocalConnectionOptions

* ExtensionParameters

* ConnectionModes

* Events

* Signals

* Actions

* DigitMapLetters

* ConnectionParameters

* RestartMethods

* ReasonCodes

* Return codes

each of which will be explained in more detail below. The rules for defining each of these extensions in a package are described in Section 6, and the encoding and syntax are defined in Section 3 and Appendix A.

With the exception of DigitMapLetters, a package defines a separate name space for each type of extension by adding the package name as a prefix to the extension, i.e.:

   package-name/extension

Thus the package-name is followed by a slash ("/") and the name of the extension.

An endpoint supporting one or more packages may define one of those packages as the default package for the endpoint. Use of the package name for events and signals in the default package for an endpoint is OPTIONAL, however it is RECOMMENDED to always include the package name. All other extensions, except DigitMapLetter, defined in the package MUST include the package-name when referring to the extension.

Package names are case insensitive strings of letters, hyphens and digits, with the restriction that hyphens shall never be the first or last character in a name. Examples of package names are "D", "T", and "XYZ". Package names are not case sensitive - names such as "XYZ", "xyz", and "xYz" are equal.

Package definitions will be provided in other documents and with package names and extensions names registered with IANA. For more details, refer to section 6.

Implementers can gain experience by using experimental packages. The name of an experimental package MUST start with the two characters "x-"; the IANA SHALL NOT register package names that start with these characters, or the characters "x+", which are reserved. A gateway that receives a command referring to an unsupported package MUST return an error (error code 518 - unsupported package, is RECOMMENDED).

2.1.7 Events and Signals

The concept of events and signals is central to MGCP. A Call Agent may ask to be notified about certain events occurring in an endpoint (e.g., off-hook events) by including the name of the event in a RequestedEvents parameter (in a NotificationRequest command - see Section 2.3.3).

A Call Agent may also request certain signals to be applied to an endpoint (e.g., dial-tone) by supplying the name of the event in a SignalRequests parameter.

Events and signals are grouped in packages, within which they share the same name space which we will refer to as event names in the following. Event names are case insensitive strings of letters, hyphens and digits, with the restriction that hyphens SHALL NOT be the first or last character in a name. Some event codes may need to be parameterized with additional data, which is accomplished by adding the parameters between a set of parentheses. Event names are not case sensitive - values such as "hu", "Hu", "HU" or "hU" are equal.

Examples of event names can be "hu" (off hook or "hang-up" transition), "hf" (hook-flash) or "0" (the digit zero).

The package name is OPTIONAL for events in the default package for an endpoint, however it is RECOMMENDED to always include the package name. If the package name is excluded from the event name, the default package name for that endpoint MUST be assumed. For example, for an analog access line which has the line package ("L") as a default with dial-tone ("dl") as one of the events in that package, the following two event names are equal:

L/dl

and

dl

For any other non-default packages that are associated with that endpoint, (such as the generic package for an analog access endpoint-type for example), the package name MUST be included with the event name. Again, unconditional inclusion of the package name is RECOMMENDED.

Digits, or letters, are supported in some packages, notably "DTMF". Digits and letters are defined by the rules "Digit" and "Letter" in the definition of digit maps. This definition refers to the digits (0 to 9), to the asterisk or star ("*") and orthotrope, number or pound sign ("#"), and to the letters "A", "B", "C" and "D", as well as the timer indication "T". These letters can be combined in "digit string" that represents the keys that a user punched on a dial. In addition, the letter "X" can be used to represent all digits (0 to 9). Also, extensions MAY define use of other letters. The need to easily express the digit strings in earlier versions of the protocol has a consequence on the form of event names:

An event name that does not denote a digit MUST always contain at least one character that is neither a digit, nor one of the letters A, B, C, D, T or X (such names also MUST NOT just contain the special signs "*", or "#"). Event names consisting of more than one character however may use any of the above.

A Call Agent may often have to ask a gateway to detect a group of events. Two conventions can be used to denote such groups:

* The "*" and "all" wildcard conventions (see below) can be used to detect any event belonging to a package, or a given event in many packages, or any event in any package supported by the gateway.

* The regular expression Range notation can be used to detect a range of digits.

The star sign (*) can be used as a wildcard instead of a package name, and the keyword "all" can be used as a wildcard instead of an event name:

* A name such as "foo/all" denotes all events in package "foo".

* A name such as "*/bar" denotes the event "bar" in any package supported by the gateway.

* The name "*/all" denotes all events supported by the endpoint.

This specification purposely does not define any additional detail for the "all packages" and "all events" wildcards. They provide limited benefits, but introduce significant complexity along with the potential for errors. Their use is consequently strongly discouraged.

The Call Agent can ask a gateway to detect a set of digits or letters either by individually describing those letters, or by using the "range" notation defined in the syntax of digit strings. For example, the Call Agent can:

* Use the letter "x" to denote" digits from 0 to 9.
* Use the notation "[0-9#]" to denote the digits 0 to 9 and the pound sign.

The individual event codes are still defined in a package though (e.g., the "DTMF" package).

Events can by default only be generated and detected on endpoints, however events can be also be defined so they can be generated or detected on connections rather than on the endpoint itself (see Section 6.6). For example, gateways may be asked to provide a ringback tone on a connection. When an event is to be applied on a connection, the name of the connection MUST be added to the name of the event, using an "at" sign (@) as a delimiter, as in:

```
G/rt@0A3F58
```

where "G" is the name of the package and "rt" is the name of the event. Should the connection be deleted while an event or signal is being detected or applied on it, that particular event detection or signal generation simply stops. Depending on the signal, this may generate a failure (see below).

The wildcard character "*" (star) can be used to denote "all connections". When this convention is used, the gateway will generate or detect the event on all the connections that are connected to the endpoint. This applies to existing as well as future connections created on the endpoint. An example of this convention could be:

```
R/qa@*
```

where "R" is the name of the package and "qa" is the name of the event.

When processing a command using the "all connections" wildcard, the "*" wildcard character applies to all current and future connections on the endpoint, however it will not be expanded. If a subsequent command either explicitly (e.g., by auditing) or implicitly (e.g., by persistence) refers to such an event, the "*" value will be used. However, when the event is actually observed, that particular occurrence of the event will include the name of the specific connection it occurred on.

The wildcard character "$" can be used to denote "the current connection". It can only be used by the Call Agent, when the event notification request is "encapsulated" within a connection creation or modification command. When this convention is used, the gateway will generate or detect the event on the connection that is currently being created or modified. An example of this convention is:

```
G/rt@$
```

When processing a command using the "current connection" wildcard, the "$" wildcard character will be expanded to the value of the current connection. If a subsequent command either explicitly (e.g., by auditing) or implicitly (e.g., by persistence) refers to such an event, the expanded value will be used. In other words, the "current connection" wildcard is expanded once, which is at the initial processing of the command in which it was explicitly included.

The connection id, or a wildcard replacement, can be used in conjunction with the "all packages" and "all events" conventions. For example, the notation:

```
*/all@*
```

can be used to designate all events on all current and future connections on the endpoint. However, as mentioned before, the use of the "all packages" and "all events" wildcards are strongly discouraged.

Signals are divided into different types depending on their behavior:

* On/off (OO): Once applied, these signals last until they are turned off. This can only happen as the result of a reboot/restart or a new SignalRequests where the signal is explicitly turned off (see later). Signals of type OO are defined to be idempotent, thus multiple requests to turn a given OO signal on (or off) are

perfectly valid and MUST NOT result in any errors. An On/Off signal could be a visual message-waiting indicator (VMWI). Once turned on, it MUST NOT be turned off until explicitly instructed to by the Call Agent, or as a result of an endpoint restart, i.e., these signals will not turn off as a result of the detection of a requested event.

* Time-out (TO): Once applied, these signals last until they are either cancelled (by the occurrence of an event or by not being included in a subsequent (possibly empty) list of signals), or a signal-specific period of time has elapsed. A TO signal that times out will generate an "operation complete" event. A TO signal could be "ringback" timing out after 180 seconds. If an event occurs prior to the 180 seconds, the signal will, by default, be stopped (the "Keep signals active" action - see Section 2.3.3 - will override this behavior). If the signal is not stopped, the signal will time out, stop and generate an "operation complete" event, about which the Call Agent may or may not have requested to be notified. If the Call Agent has asked for the "operation complete" event to be notified, the "operation complete" event sent to the Call Agent SHALL include the name(s) of the signal(s) that timed out (note that if parameters were passed to the signal, the parameters will not be reported). If the signal was generated on a connection, the name of the connection SHALL be included as described above. Time-out signals have a default time-out value defined for them, which MAY be altered by the provisioning process. Also, the time-out period may be provided as a parameter to the signal (see Section 3.2.2.4). A value of zero indicates that the time-out period is infinite. A TO signal that fails after being started, but before having generated an "operation complete" event will generate an "operation failure" event which will include the name of the signal that failed. Deletion of a connection with an active TO signal will result in such a failure.

* Brief (BR): The duration of these signals is normally so short that they stop on their own. If a signal stopping event occurs, or a new SignalRequests is applied, a currently active BR signal will not stop. However, any pending BR signals not yet applied MUST be cancelled (a BR signal becomes pending if a NotificationRequest includes a BR signal, and there is already an active BR signal). As an example, a brief tone could be a DTMF digit. If the DTMF digit "1" is currently being played, and a signal stopping event occurs, the "1" would play to completion. If a request to play DTMF digit "2" arrives before DTMF digit "1" finishes playing, DTMF digit "2" would become pending.

Signal(s) generated on a connection MUST include the name of that connection.

2.2 Usage of SDP

The Call Agent uses the MGCP to provide the endpoint with the description of connection parameters such as IP addresses, UDP port and RTP profiles. These descriptions will follow the conventions delineated in the Session Description Protocol which is now an IETF proposed standard, documented in RFC 2327.

2.3 Gateway Control Commands

2.3.1 Overview of Commands

This section describes the commands of the MGCP. The service consists of connection handling and endpoint handling commands. There are currently nine commands in the protocol:

* The Call Agent can issue an EndpointConfiguration command to a gateway, instructing the gateway about the coding characteristics expected by the "line-side" of the endpoint.

* The Call Agent can issue a NotificationRequest command to a gateway, instructing the gateway to watch for specific events such as hook actions or DTMF tones on a specified endpoint.

* The gateway will then use the Notify command to inform the Call Agent when the requested events occur.

* The Call Agent can use the CreateConnection command to create a connection that terminates in an "endpoint" inside the gateway.

* The Call Agent can use the ModifyConnection command to change the parameters associated with a previously established connection.

* The Call Agent can use the DeleteConnection command to delete an existing connection. The DeleteConnection command may also be used by a gateway to indicate that a connection can no longer be sustained.

* The Call Agent can use the AuditEndpoint and AuditConnection commands to audit the status of an "endpoint" and any connections associated with it. Network management beyond the capabilities provided by these commands is generally desirable. Such capabilities are expected to be supported by the use of the Simple Network Management Protocol (SNMP) and definition of a MIB which is outside the scope of this specification.

* The Gateway can use the RestartInProgress command to notify the Call Agent that a group of endpoints managed by the gateway is being taken out-of-service or is being placed back in-service.

These services allow a controller (normally, the Call Agent) to instruct a gateway on the creation of connections that terminate in an "endpoint" attached to the gateway, and to be informed about events occurring at the endpoint. An endpoint may be for example:

* A specific trunk circuit, within a trunk group terminating in a gateway,

* A specific announcement handled by an announcement server.

Connections are logically grouped into "calls" (the concept of a "call" has however little semantic meaning in MGCP itself). Several connections, that may or may not belong to the same call, can terminate in the same endpoint. Each connection is qualified by a "mode" parameter, which can be set to "send only" (sendonly), "receive only" (recvonly), "send/receive" (sendrecv), "conference" (confrnce), "inactive" (inactive), "loopback", "continuity test" (conttest), "network loop back" (netwloop) or "network continuity test" (netwtest).

Media generated by the endpoint is sent on connections whose mode is either "send only", "send/receive", or "conference", unless the endpoint has a connection in "loopback" or "continuity test" mode. However, media generated by applying a signal to a connection is always sent on the connection, regardless of the mode.

The handling of the media streams received on connections is determined by the mode parameters:

* Media streams received through connections in "receive", "conference" or "send/receive" mode are mixed and sent to the endpoint, unless the endpoint has another connection in "loopback" or "continuity test" mode.

* Media streams originating from the endpoint are transmitted over all the connections whose mode is "send", "conference" or "send/receive", unless the endpoint has another connection in "loopback" or "continuity test" mode.

* In addition to being sent to the endpoint, a media stream received through a connection in "conference" mode is forwarded to all the other connections whose mode is "conference". This also applies

when the endpoint has a connection in "loopback" or "continuity test" mode. The details of this forwarding, e.g., RTP translator or mixer, is outside the scope of this document.

Note that in order to detect events on a connection, the connection must by default be in one of the modes "receive", "conference", "send/receive", "network loopback" or "network continuity test". The event detection only applies to the incoming media. Connections in "sendonly", "inactive", "loopback", or "continuity test" mode will thus normally not detect any events, although requesting to do so is not considered an error.

The "loopback" and "continuity test" modes are used during maintenance and continuity test operations. An endpoint may have more than one connection in either "loopback" or "continuity test" mode. As long as there is one connection in that particular mode, and no other connection on the endpoint is placed in a different maintenance or test mode, the maintenance or test operation shall continue undisturbed. There are two flavors of continuity test, one specified by ITU and one used in the US. In the first case, the test is a loopback test. The originating switch will send a tone (the go tone) on the bearer circuit and expects the terminating switch to loopback the tone. If the originating switch sees the same tone returned (the return tone), the COT has passed. If not, the COT has failed. In the second case, the go and return tones are different. The originating switch sends a certain go tone. The terminating switch detects the go tone, it asserts a different return tone in the backwards direction. When the originating switch detects the return tone, the COT is passed. If the originating switch never detects the return tone, the COT has failed.

If the mode is set to "loopback", the gateway is expected to return the incoming signal from the endpoint back into that same endpoint. This procedure will be used, typically, for testing the continuity of trunk circuits according to the ITU specifications. If the mode is set to "continuity test", the gateway is informed that the other end of the circuit has initiated a continuity test procedure according to the GR specification (see [22]). The gateway will place the circuit in the transponder mode required for dual-tone continuity tests.

If the mode is set to "network loopback", the audio signals received from the connection will be echoed back on the same connection. The media is not forwarded to the endpoint.

If the mode is set to "network continuity test", the gateway will process the packets received from the connection according to the transponder mode required for dual-tone continuity test, and send the processed signal back on the connection. The media is not forwarded

to the endpoint. The "network continuity test" mode is included for backwards compatibility only and use of it is discouraged.

2.3.2 EndpointConfiguration

The EndpointConfiguration command can be used to specify the encoding of the signals that will be received by the endpoint. For example, in certain international telephony configurations, some calls will carry mu-law encoded audio signals, while others will use A-law. The Call Agent can use the EndpointConfiguration command to pass this information to the gateway. The configuration may vary on a call by call basis, but can also be used in the absence of any connection.

```
ReturnCode,
[PackageList]
<-- EndpointConfiguration(EndpointId,
                          [BearerInformation])
```

EndpointId is the name of the endpoint(s) in the gateway where EndpointConfiguration executes. The "any of" wildcard convention MUST NOT be used. If the "all of" wildcard convention is used, the command applies to all the endpoints whose name matches the wildcard.

BearerInformation is a parameter defining the coding of the data sent to and received from the line side. The information is encoded as a list of sub-parameters. The only sub-parameter defined in this version of the specification is the bearer encoding, whose value can be set to "A-law" or "mu-law". The set of sub-parameters may be extended.

In order to allow for extensibility, while remaining backwards compatible, the BearerInformation parameter is conditionally optional based on the following conditions:

* if Extension Parameters (vendor, package or other) are not used, the BearerInformation parameter is REQUIRED,

* otherwise, the BearerInformation parameter is OPTIONAL.

When omitted, BearerInformation MUST retain its current value.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.3 NotificationRequest

The NotificationRequest command is used to request the gateway to send notifications upon the occurrence of specified events in an endpoint. For example, a notification may be requested for when a gateway detects that an endpoint is receiving tones associated with fax communication. The entity receiving this notification may then decide to specify use of a different type of encoding method in the connections bound to this endpoint and instruct the gateway accordingly with a ModifyConnection Command.

```
      ReturnCode,
      [PackageList]
      <-- NotificationRequest(EndpointId,
                              [NotifiedEntity,]
                              [RequestedEvents,]
                              RequestIdentifier,
                              [DigitMap,]
                              [SignalRequests,]
                              [QuarantineHandling,]
                              [DetectEvents,]
                              [encapsulated EndpointConfiguration])
```

EndpointId is the identifier for the endpoint(s) in the the gateway where the NotificationRequest executes. The "any of" wildcard MUST NOT be used.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint.

RequestIdentifier is used to correlate this request with the notifications that it triggers. It will be repeated in the corresponding Notify command.

RequestedEvents is a list of events, possibly qualified by event parameters (see Section 3.2.2.4), that the gateway is requested to detect and report. Such events may include, for example, fax tones, continuity tones, or on-hook transition. Unless otherwise specified, events are detected on the endpoint, however some events can be detected on a connection. A given event MUST NOT appear more than once in a RequestedEvents. If the parameter is omitted, it defaults to empty.

To each event is associated one or more actions, which can be:

* Notify the event immediately, together with the accumulated list of observed events,

* Swap audio,

* Accumulate the event in an event buffer, but don't notify yet,

* Accumulate according to Digit Map,

* Keep Signal(s) active,

* Process the Embedded Notification Request,

* Ignore the event.

Support for Notify, Accumulate, Keep Signal(s) Active, Embedded Notification Request, and Ignore is REQUIRED. Support for Accumulate according to Digit Map is REQUIRED on any endpoint capable of detecting DTMF. Support for any other action is OPTIONAL. The set of actions can be extended.

A given action can by default be specified for any event, although some actions will not make sense for all events. For example, an off-hook event with the Accumulate according to Digit Map action is valid, but will of course immediately trigger a digit map mismatch when the off-hook event occurs. Needless to say, such practice is discouraged.

Some actions can be combined as shown in the table below, where "Y" means the two actions can be combined, and "N" means they cannot:

	Notif	Swap	Accum	AccDi	KeSiA	EmbNo	Ignor
Notif	N	Y	N	N	Y	Y*	N
Swap	-	N	Y	N	N	N	Y
Accum	-	-	N	N	Y	Y	N
AccDi	-	-	-	N	Y	N	N
KeSiA	-	-	-	-	N	Y	Y
EmbNo	-	-	-	-	-	N	N
Ignor	-	-	-	-	-	-	N

Note (*): The "Embedded Notification Request" can only be combined with "Notify", if the gateway is allowed to issue more than one Notify command per Notification request (see below and Section 4.4.1).

If no action is specified, the Notify action will be applied. If one or more actions are specified, only those actions apply. When two or more actions are specified, each action MUST be combinable with all

the other actions as defined by the table above - the individual actions are assumed to occur simultaneously.

If a client receives a request with an invalid or unsupported action or an illegal combination of actions, it MUST return an error to the Call Agent (error code 523 - unknown or illegal combination of actions, is RECOMMENDED).

In addition to the RequestedEvents parameter specified in the command, some MGCP packages may contain "persistent events" (this is generally discouraged though - see Appendix B for an alternative). Persistent events in a given package are always detected on an endpoint that implements that package. If a persistent event is not included in the list of RequestedEvents, and the event occurs, the event will be detected anyway and processed like all other events, as if the persistent event had been requested with a Notify action. A NotificationRequest MUST still be in place for a persistent event to trigger a Notify though. Thus, informally, persistent events can be viewed as always being implicitly included in the list of RequestedEvents with an action to Notify, although no glare detection, etc., will be performed.

Non-persistent events are those events that need to be explicitly included in the RequestedEvents list. The (possibly empty) list of requested events completely replaces the previous list of requested events. In addition to the persistent events, only the events specified in the requested events list will be detected by the endpoint. If a persistent event is included in the RequestedEvents list, the action specified will replace the default action associated with the event for the life of the RequestedEvents list, after which the default action is restored. For example, if "off-hook"was a persistent event, the "Ignore off-hook" action was specified, and a new request without any off-hook instructions were received, the default "Notify off-hook" operation would be restored.

The gateway will detect the union of the persistent events and the requested events. If an event is not included in either list, it will be ignored.

The Call Agent can send a NotificationRequest with an empty (or omitted) RequestedEvents list to the gateway. The Call Agent can do so, for example, to a gateway when it does not want to collect any more DTMF digits. However, persistent events will still be detected and notified.

The Swap Audio action can be used when a gateway handles more than one connection on an endpoint. This will be the case for call waiting, and possibly other feature scenarios. In order to avoid the

round-trip to the Call Agent when just changing which connection is attached to the audio functions of the endpoint, the NotificationRequest can map an event (usually hook flash, but could be some other event) to a local swap audio function, which selects the "next" connection in a round robin fashion. If there is only one connection, this action is effectively a no-op. If there are more than two connections, the order is undefined. If the endpoint has exactly two connections, one of which is "inactive", the other of which is in "send/receive" mode, then swap audio will attempt to make the "send/receive" connection "inactive", and vice versa. This specification intentionally does not provide any additional detail on the swap audio action.

If signal(s) are desired to start when an event being looked for occurs, the "Embedded NotificationRequest" action can be used. The embedded NotificationRequest may include a new list of RequestedEvents, SignalRequests and a new digit map as well. The semantics of the embedded NotificationRequest is as if a new NotificationRequest was just received with the same NotifiedEntity, RequestIdentifier, QuarantineHandling and DetectEvents. When the "Embedded NotificationRequest" is activated, the "current dial string" will be cleared; however the list of observed events and the quarantine buffer will be unaffected (if combined with a Notify, the Notify will clear the list of observed events though - see Section 4.4.1). Note, that the Embedded NotificationRequest action does not accumulate the triggering event, however it can be combined with the Accumulate action to achieve that. If the Embedded NotificationRequest fails, an Embedded NotificationRequest failure event SHOULD be generated (see Appendix B).

MGCP implementations SHALL be able to support at least one level of embedding. An embedded NotificationRequest that respects this limitation MUST NOT contain another Embedded NotificationRequest.

DigitMap is an optional parameter that allows the Call Agent to provision the endpoint with a digit map according to which digits will be accumulated. If this optional parameter is absent, the previously defined value is retained. This parameter MUST be defined, either explicitly or through a previous command, if the RequestedEvents parameter contains a request to "accumulate according to the digit map". The collection of these digits will result in a digit string. The digit string is initialized to a null string upon reception of the NotificationRequest, so that a subsequent notification only returns the digits that were collected after this request. Digits that were accumulated according to the digit map are reported as any other accumulated event, in the order in which they occur. It is therefore possible that other events accumulated are

found in between the list of digits. If the gateway is requested to "accumulate according to digit map" and the gateway currently does not have a digit map for the endpoint in question, the gateway MUST return an error (error code 519 - endpoint does not have a digit map, is RECOMMENDED).

SignalRequests is an optional parameter that contains the set of signals that the gateway is asked to apply. When omitted, it defaults to empty. When multiple signals are specified, the signals MUST be applied in parallel. Unless otherwise specified, signals are applied to the endpoint. However some signals can be applied to a connection. Signals are identified by their name, which is an event name, and may be qualified by signal parameters (see Section 3.2.2.4). The following are examples of signals:

* Ringing,

* Busy tone,

* Call waiting tone,

* Off hook warning tone,

* Ringback tones on a connection.

Names and descriptions of signals are defined in the appropriate package.

Signals are, by default, applied to endpoints. If a signal applied to an endpoint results in the generation of a media stream (audio, video, etc.), then by default the media stream MUST NOT be forwarded on any connection associated with that endpoint, regardless of the mode of the connection. For example, if a call-waiting tone is applied to an endpoint involved in an active call, only the party using the endpoint in question will hear the call-waiting tone. However, individual signals may define a different behavior.

When a signal is applied to a connection that has received a RemoteConnectionDescriptor, the media stream generated by that signal will be forwarded on the connection regardless of the current mode of the connection (including loopback and continuity test). If a RemoteConnectionDescriptor has not been received, the gateway MUST return an error (error code 527 - missing RemoteConnectionDescriptor, is RECOMMENDED). Note that this restriction does not apply to detecting events on a connection.

When a (possibly empty) list of signal(s) is supplied, this list completely replaces the current list of active time-out signals. Currently active time-out signals that are not provided in the new list MUST be stopped and the new signal(s) provided will now become active. Currently active time-out signals that are provided in the new list of signals MUST remain active without interruption, thus the timer for such time-out signals will not be affected. Consequently, there is currently no way to restart the timer for a currently active time-out signal without turning the signal off first. If the time-out signal is parameterized, the original set of parameters MUST remain in effect, regardless of what values are provided subsequently. A given signal MUST NOT appear more than once in a SignalRequests. Note that applying a signal S to an endpoint, connection C1 and connection C2, constitutes three different and independent signals.

The action triggered by the SignalRequests is synchronized with the collection of events specified in the RequestedEvents parameter. For example, if the NotificationRequest mandates "ringing" and the RequestedEvents asks to look for an "off-hook" event, the ringing SHALL stop as soon as the gateway detects an off-hook event. The formal definition is that the generation of all "Time Out" signals SHALL stop as soon as one of the requested events is detected, unless the "Keep signals active" action is associated to the detected event. The RequestedEvents and SignalRequests may refer to the same event definitions. In one case, the gateway is asked to detect the occurrence of the event, and in the other case it is asked to generate it. The specific events and signals that a given endpoint can detect or perform are determined by the list of packages that are supported by that endpoint. Each package specifies a list of events and signals that can be detected or performed. A gateway that is requested to detect or perform an event belonging to a package that is not supported by the specified endpoint MUST return an error (error code 518 - unsupported or unknown package, is RECOMMENDED). When the event name is not qualified by a package name, the default package name for the endpoint is assumed. If the event name is not registered in this default package, the gateway MUST return an error (error code 522 - no such event or signal, is RECOMMENDED).

The Call Agent can send a NotificationRequest whose requested signal list is empty. It will do so for example when a time-out signal(s) should stop.

If signal(s) are desired to start as soon as a "looked-for" event occurs, the "Embedded NotificationRequest" action can be used. The embedded NotificationRequest may include a new list of RequestedEvents, SignalRequests and a new Digit Map as well. The embedded NotificationRequest action allows the Call Agent to set up a

"mini-script" to be processed by the gateway immediately following the detection of the associated event. Any SignalRequests specified in the embedded NotificationRequest will start immediately. Considerable care must be taken to prevent discrepancies between the Call Agent and the gateway. However, long-term discrepancies should not occur as a new SignalRequests completely replaces the old list of active time-out signals, and BR-type signals always stop on their own. Limiting the number of On/Off-type signals is encouraged. It is considered good practice for a Call Agent to occasionally turn on all On/Off signals that should be on, and turn off all On/Off signals that should be off.

The Ignore action can be used to ignore an event, e.g., to prevent a persistent event from being notified. However, the synchronization between the event and an active time-out signal will still occur by default (e.g., a time-out dial-tone signal will stop when an off-hook occurs even if off-hook was a requested event with action "Ignore"). To prevent this synchronization from happening, the "Keep Signal(s) Active" action will have to be specified as well.

The optional QuarantineHandling parameter specifies the handling of "quarantine" events, i.e., events that have been detected by the gateway before the arrival of this NotificationRequest command, but have not yet been notified to the Call Agent. The parameter provides a set of handling options (see Section 4.4.1 for details):

* whether the quarantined events should be processed or discarded (the default is to process them).

* whether the gateway is expected to generate at most one notification (step by step), or multiple notifications (loop), in response to this request (the default is at most one).

When the parameter is absent, the default value is assumed.

We should note that the quarantine-handling parameter also governs the handling of events that were detected and processed but not yet notified when the command is received.

DetectEvents is an optional parameter, possibly qualified by event parameters, that specifies a list of events that the gateway is requested to detect during the quarantine period. When this parameter is absent, the events to be detected in the quarantine period are those listed in the last received DetectEvents list. In addition, the gateway will also detect persistent events and the events specified in the RequestedEvents list, including those for which the "ignore" action is specified.

Some events and signals, such as the in-line ringback or the quality alert, are performed or detected on connections terminating in the endpoint rather than on the endpoint itself. The structure of the event names (see Section 2.1.7) allows the Call Agent to specify the connection(s) on which the events should be performed or detected.

The NotificationRequest command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint(s). When this command is present, the parameters of the EndpointConfiguration command are included with the normal parameters of the NotificationRequest, with the exception of the EndpointId, which is not replicated.

The encapsulated EndpointConfiguration command shares the fate of the NotificationRequest command. If the NotificationRequest is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.4 Notify

Notifications with the observed events are sent by the gateway via the Notify command when a triggering event occurs.

```
      ReturnCode,
      [PackageList]
      <-- Notify(EndpointId,
                 [NotifiedEntity,]
                 RequestIdentifier,
                 ObservedEvents)
```

EndpointId is the name for the endpoint in the gateway which is issuing the Notify command. The identifier MUST be a fully qualified endpoint identifier, including the domain name of the gateway. The local part of the name MUST NOT use any of the wildcard conventions.

NotifiedEntity is a parameter that identifies the entity which requested the notification. This parameter is equal to the NotifiedEntity parameter of the NotificationRequest that triggered this notification. The parameter is absent if there was no such parameter in the triggering request. Regardless of the value of the NotifiedEntity parameter, the notification MUST be sent to the current "notified entity" for the endpoint.

RequestIdentifier is a parameter that repeats the RequestIdentifier parameter of the NotificationRequest that triggered this notification. It is used to correlate this notification with the request that triggered it. Persistent events will be viewed here as if they had been included in the last NotificationRequest. An implicit NotificationRequest MAY be in place right after restart - the RequestIdentifier used for it will be zero ("0") - see Section 4.4.1 for details.

ObservedEvents is a list of events that the gateway detected and accumulated. A single notification may report a list of events that will be reported in the order in which they were detected (FIFO).

The list will only contain the identification of events that were requested in the RequestedEvents parameter of the triggering NotificationRequest. It will contain the events that were either accumulated (but not notified) or treated according to digit map (but no match yet), and the final event that triggered the notification or provided a final match in the digit map. It should be noted that digits MUST be added to the list of observed events as they are accumulated, irrespective of whether they are accumulated according to the digit map or not. For example, if a user enters the digits "1234" and some event E is accumulated between the digits "3" and "4" being entered, the list of observed events would be "1, 2, 3, E, 4". Events that were detected on a connection SHALL include the name of that connection as in "R/qa@0A3F58" (see Section 2.1.7).

If the list of ObservedEvents reaches the capacity of the endpoint, an ObservedEvents Full event (see Appendix B) SHOULD be generated (the endpoint shall ensure it has capacity to include this event in the list of ObservedEvents). If the ObservedEvents Full event is not used to trigger a Notify, event processing continues as before (including digit map matching); however, the subsequent events will not be included in the list of ObservedEvents.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.5 CreateConnection

This command is used to create a connection between two endpoints.

```
      ReturnCode,
      [ConnectionId,]
      [SpecificEndPointId,]
      [LocalConnectionDescriptor,]
      [SecondEndPointId,]
      [SecondConnectionId,]
      [PackageList]
      <-- CreateConnection(CallId,
                           EndpointId,
                           [NotifiedEntity,]
                           [LocalConnectionOptions,]
                           Mode,
                           [{RemoteConnectionDescriptor |
                           SecondEndpointId}, ]
                           [Encapsulated NotificationRequest,]
                           [Encapsulated EndpointConfiguration])
```

A connection is defined by its endpoints. The input parameters in CreateConnection provide the data necessary to build a gateway's "view" of a connection.

CallId is a parameter that identifies the call (or session) to which this connection belongs. This parameter SHOULD, at a minimum, be unique within the collection of Call Agents that control the same gateways. Connections that belong to the same call SHOULD share the same call-id. The call-id has little semantic meaning in the protocol; however it can be used to identify calls for reporting and accounting purposes. It does not affect the handling of connections by the gateway.

EndpointId is the identifier for the connection endpoint in the gateway where CreateConnection executes. The EndpointId can be fully-specified by assigning a value to the parameter EndpointId in the function call or it may be under-specified by using the "any of" wildcard convention. If the endpoint is underspecified, the endpoint identifier SHALL be assigned by the gateway and its complete value returned in the SpecificEndPointId parameter of the response. When the "any of" wildcard is used, the endpoint assigned MUST be in-service and MUST NOT already have any connections on it. If no such endpoint is available, error code 410 (no endpoint available) SHOULD be returned. The "all of" wildcard MUST NOT be used.

The NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint.

LocalConnectionOptions is an optional structure used by the Call Agent to direct the handling of the connection by the gateway. The fields contained in a LocalConnectionOptions structure may include one or more of the following (each field MUST NOT be supplied more than once):

* Codec compression algorithm: One or more codecs, listed in order of preference. For interoperability, it is RECOMMENDED to support G.711 mu-law encoding ("PCMU"). See Section 2.6 for details on the codec selection process.

* Packetization period: A single millisecond value or a range may be specified. The packetization period SHOULD NOT contradict the specification of the codec compression algorithm. If a codec is specified that has a frame size which is inconsistent with the packetization period, and that codec is selected, the gateway is authorized to use a packetization period that is consistent with the frame size even if it is different from that specified. In so doing, the gateway SHOULD choose a non-zero packetization period as close to that specified as possible. If a packetization period is not specified, the endpoint SHOULD use the default packetization period(s) for the codec(s) selected.

* Bandwidth: The allowable bandwidth, i.e., payload plus any header overhead from the transport layer and up, e.g., IP, UDP, and RTP. The bandwidth specification SHOULD NOT contradict the specification of codec compression algorithm or packetization period. If a codec is specified, then the gateway is authorized to use it, even if it results in the usage of a larger bandwidth than specified. Any discrepancy between the bandwidth and codec specification will not be reported as an error.

* Type of Service: This indicates the class of service to be used for this connection. When the Type of Service is not specified, the gateway SHALL use a default value of zero unless provisioned otherwise.

* Usage of echo cancellation: By default, the telephony gateways always perform echo cancellation on the endpoint. However, it may be necessary, for some calls, to turn off these operations. The echo cancellation parameter can have two values, "on" (when the echo cancellation is requested) and "off" (when it is turned off). The parameter is optional. If the parameter is omitted when creating a connection and there are no other connections on the endpoint, the endpoint SHALL apply echo cancellation initially. If the parameter is omitted when creating a connection and there are existing connections on the endpoint, echo cancellation is unchanged. The endpoint SHOULD subsequently enable or disable echo

cancellation when voiceband data is detected - see e.g., ITU-T recommendation V.8, V.25, and G.168. Following termination of voiceband data, the handling of echo cancellation SHALL then revert to the current value of the echo cancellation parameter. It is RECOMMENDED that echo cancellation handling is left to the gateway rather than having this parameter specified by the Call Agent.

* Silence Suppression: The telephony gateways may perform voice activity detection, and avoid sending packets during periods of silence. However, it is necessary, for example for modem calls, to turn off this detection. The silence suppression parameter can have two values, "on" (when the detection is requested) and "off" (when it is not requested). The default is "off" (unless provisioned otherwise). Upon detecting voiceband data, the endpoint SHOULD disable silence suppression. Following termination of voiceband data, the handling of silence suppression SHALL then revert to the current value of the silence suppression parameter.

* Gain Control: The telephony gateways may perform gain control on the endpoint, in order to adapt the level of the signal. However, it is necessary, for example for some modem calls, to turn off this function. The gain control parameter may either be specified as "automatic", or as an explicit number of decibels of gain. The gain specified will be added to media sent out over the endpoint (as opposed to the connection) and subtracted from media received on the endpoint. The parameter is optional. When there are no other connections on the endpoint, and the parameter is omitted, the default is to not perform gain control (unless provisioned otherwise), which is equivalent to specifying a gain of 0 decibels. If there are other connections on the endpoint, and the parameter is omitted, gain control is unchanged. Upon detecting voiceband data, the endpoint SHOULD disable gain control if needed. Following termination of voiceband data, the handling of gain control SHALL then revert to the current value of the gain control parameter. It should be noted, that handling of gain control is normally best left to the gateway and hence use of this parameter is NOT RECOMMENDED.

* RTP security: The Call agent can request the gateway to enable encryption of the audio Packets. It does so by providing a key specification, as specified in RFC 2327. By default, encryption is not performed.

* Network Type: The Call Agent may instruct the gateway to prepare the connection on a specified type of network. If absent, the value is based on the network type of the gateway being used.

* Resource reservation: The Call Agent may instruct the gateway to use network resource reservation for the connection. See Section 2.7 for details.

The Call Agent specifies the relevant fields it cares about in the command and leaves the rest to the discretion of the gateway. For those of the above parameters that were not explicitly included, the gateway SHOULD use the default values if possible. For a detailed list of local connection options included with this specification refer to section 3.2.2.10. The set of local connection options can be extended.

The Mode indicates the mode of operation for this side of the connection. The basic modes are "send", "receive", "send/receive", "conference", "inactive", "loopback", "continuity test", "network loop back" and "network continuity test". The expected handling of these modes is specified in the introduction of the "Gateway Control Commands", Section 2.3. Note that signals applied to a connection do not follow the connection mode. Some endpoints may not be capable of supporting all modes. If the command specifies a mode that the endpoint does not support, an error SHALL be returned (error 517 - unsupported mode, is RECOMMENDED). Also, if a connection has not yet received a RemoteConnectionDescriptor, an error MUST be returned if the connection is attempted to be placed in any of the modes "send only", "send/receive", "conference", "network loopback", "network continuity test", or if a signal (as opposed to detecting an event) is to be applied to the connection (error code 527 - missing RemoteConnectionDescriptor, is RECOMMENDED). The set of modes can be extended.

The gateway returns a ConnectionId, that uniquely identifies the connection within the endpoint, and a LocalConnectionDescriptor, which is a session description that contains information about the connection, e.g., IP address and port for the media, as defined in SDP.

The SpecificEndPointId is an optional parameter that identifies the responding endpoint. It is returned when the EndpointId argument referred to an "any of" wildcard name and the command succeeded. When a SpecificEndPointId is returned, the Call Agent SHALL use it as the EndpointId value in successive commands referring to this connection.

The SecondEndpointId can be used instead of the RemoteConnectionDescriptor to establish a connection between two endpoints located on the same gateway. The connection is by definition a local connection. The SecondEndpointId can be fully-specified by assigning a value to the parameter SecondEndpointId in

the function call or it may be under-specified by using the "any of" wildcard convention. If the SecondEndpointId is underspecified, the second endpoint identifier will be assigned by the gateway and its complete value returned in the SecondEndPointId parameter of the response.

When a SecondEndpointId is specified, the command really creates two connections that can be manipulated separately through ModifyConnection and DeleteConnection commands. In addition to the ConnectionId and LocalConnectionDescriptor for the first connection, the response to the creation provides a SecondConnectionId parameter that identifies the second connection. The second connection is established in "send/receive" mode.

After receiving a "CreateConnection" request that did not include a RemoteConnectionDescriptor parameter, a gateway is in an ambiguous situation. Because it has exported a LocalConnectionDescriptor parameter, it can potentially receive packets. Because it has not yet received the RemoteConnectionDescriptor parameter of the other gateway, it does not know whether the packets that it receives have been authorized by the Call Agent. It must thus navigate between two risks, i.e., clipping some important announcements or listening to insane data. The behavior of the gateway is determined by the value of the Mode parameter:

* If the mode was set to ReceiveOnly, the gateway MUST accept the media and transmit them through the endpoint.

* If the mode was set to Inactive, Loopback, or Continuity Test, the gateway MUST NOT transmit the media through to the endpoint.

Note that the mode values SendReceive, Conference, SendOnly, Network Loopback and Network Continuity Test do not make sense in this situation. They MUST be treated as errors, and the command MUST be rejected (error code 527 - missing RemoteConnectionDescriptor, is RECOMMENDED).

The command may optionally contain an encapsulated Notification Request command, which applies to the EndpointId, in which case a RequestIdentifier parameter MUST be present, as well as, optionally, other parameters of the NotificationRequest with the exception of the EndpointId, which is not replicated. The encapsulated NotificationRequest is executed simultaneously with the creation of the connection. For example, when the Call Agent wants to initiate a call to a residential gateway, it could:

* ask the residential gateway to prepare a connection, in order to be sure that the user can start speaking as soon as the phone goes off hook,

* ask the residential gateway to start ringing,

* ask the residential gateway to notify the Call Agent when the phone goes off-hook.

This can be accomplished in a single CreateConnection command, by also transmitting the RequestedEvents parameters for the off-hook event, and the SignalRequests parameter for the ringing signal.

When these parameters are present, the creation and the NotificationRequest MUST be synchronized, which means that both MUST be accepted, or both MUST be refused. In our example, the CreateConnection may be refused if the gateway does not have sufficient resources, or cannot get adequate resources from the local network access, and the off-hook NotificationRequest can be refused in the glare condition, if the user is already off-hook. In this example, the phone must not ring if the connection cannot be established, and the connection must not be established if the user is already off-hook.

The NotifiedEntity parameter, if present, defines the new "notified entity" for the endpoint.

The command may carry an encapsulated EndpointConfiguration command, which applies to the EndpointId. When this command is present, the parameters of the EndpointConfiguration command are included with the normal parameters of the CreateConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command. Note that both of these apply to the EndpointId only.

The encapsulated EndpointConfiguration command shares the fate of the CreateConnection command. If the CreateConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.6 ModifyConnection

This command is used to modify the characteristics of a gateway's "view" of a connection. This "view" of the call includes both the local connection descriptor as well as the remote connection descriptor.

```
      ReturnCode,
      [LocalConnectionDescriptor,]
      [PackageList]
      <-- ModifyConnection(CallId,
                           EndpointId,
                           ConnectionId,
                           [NotifiedEntity,]
                           [LocalConnectionOptions,]
                           [Mode,]
                           [RemoteConnectionDescriptor,]
                           [Encapsulated NotificationRequest,]
                           [Encapsulated EndpointConfiguration])
```

The parameters used are the same as in the CreateConnection command, with the addition of a ConnectionId that identifies the connection within the endpoint. This parameter was returned by the CreateConnection command, in addition to the local connection descriptor. It uniquely identifies the connection within the context of the endpoint. The CallId used when the connection was created MUST be included as well.

The EndpointId MUST be a fully qualified endpoint identifier. The local name MUST NOT use the wildcard conventions.

The ModifyConnection command can be used to affect parameters of a connection in the following ways:

* Provide information about the other end of the connection, through the RemoteConnectionDescriptor. If the parameter is omitted, it retains its current value.

* Activate or deactivate the connection, by changing the value of the Mode parameter. This can occur at any time during the connection, with arbitrary parameter values. If the parameter is omitted, it retains its current value.

* Change the parameters of the connection through the LocalConnectionOptions, for example by switching to a different coding scheme, changing the packetization period, or modifying the handling of echo cancellation. If one or more LocalConnectionOptions parameters are omitted, then the gateway

SHOULD refrain from changing that parameter from its current value, unless another parameter necessitating such a change is explicitly provided. For example, a codec change might require a change in silence suppression. Note that if a RemoteConnectionDescriptor is supplied, then only the LocalConnectionOptions actually supplied with the ModifyConnection command will affect the codec negotiation (as described in Section 2.6).

Connections can only be fully activated if the RemoteConnectionDescriptor has been provided to the gateway. The receive-only mode, however, can be activated without the provision of this descriptor.

The command will only return a LocalConnectionDescriptor if the local connection parameters, such as RTP ports, were modified. Thus, if, for example, only the mode of the connection is changed, a LocalConnectionDescriptor will not be returned. Note however, that inclusion of LocalConnectionOptions in the command is not a prerequisite for local connection parameter changes to occur. If a connection parameter is omitted, e.g., silence suppression, the old value of that parameter will be retained if possible. If a parameter change necessitates a change in one or more unspecified parameters, the gateway is free to choose suitable values for the unspecified parameters that must change. This can for instance happen if the packetization period was not specified. If the new codec supported the old packetization period, the value of this parameter would not change, as a change would not be necessary. However, if it did not support the old packetization period, it would choose a suitable value.

The command may optionally contain an encapsulated Notification Request command, in which case a RequestIdentifier parameter MUST be present, as well as, optionally, other parameters of the NotificationRequest with the exception of the EndpointId, which is not replicated. The encapsulated NotificationRequest is executed simultaneously with the modification of the connection. For example, when a connection is accepted, the calling gateway should be instructed to place the circuit in send-receive mode and to stop providing ringing tones. This can be accomplished in a single ModifyConnection command, by also transmitting the RequestedEvents parameters, for the on-hook event, and an empty SignalRequests parameter, to stop the provision of ringing tones.

When these parameters are present, the modification and the NotificationRequest MUST be synchronized, which means that both MUST be accepted, or both MUST be refused.

The NotifiedEntity parameter, if present, defines the new "notified entity" for the endpoint.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are included with the normal parameters of the ModifyConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the ModifyConnection command. If the ModifyConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.7 DeleteConnection (from the Call Agent)

This command is used to terminate a connection. As a side effect, it collects statistics on the execution of the connection.

```
      ReturnCode,
      ConnectionParameters,
      [PackageList]
      <-- DeleteConnection(CallId,
                           EndpointId,
                           ConnectionId,
                           [NotifiedEntity,]
                           [Encapsulated NotificationRequest,]
                           [Encapsulated EndpointConfiguration])
```

The endpoint identifier, in this form of the DeleteConnection command, SHALL be fully qualified. Wildcard conventions SHALL NOT be used.

The ConnectionId identifies the connection to be deleted. The CallId used when the connection was created is included as well.

The NotifiedEntity parameter, if present, defines the new "notified entity" for the endpoint.

In the case of IP multicast, connections can be deleted individually and independently. However, in the unicast case where a connection has two ends, a DeleteConnection command has to be sent to both gateways involved in the connection. After the connection has been deleted, media streams previously supported by the connection are no longer available. Any media packets received for the old connection are simply discarded and no new media packets for the stream are sent.

After the connection has been deleted, any loopback that has been requested for the connection must be cancelled (unless the endpoint has another connection requesting loopback).

In response to the DeleteConnection command, the gateway returns a list of connection parameters that describe statistics for the connection.

When the connection was for an Internet media stream, these parameters are:

Number of packets sent:

The total number of media packets transmitted by the sender since starting transmission on this connection. In the case of RTP, the count is not reset if the sender changes its synchronization source identifier (SSRC, as defined in RTP), for example as a result of a ModifyConnection command. The value is zero if the connection was always set in "receive only" mode and no signals were applied to the connection.

Number of octets sent:

The total number of payload octets (i.e., not including header or padding) transmitted in media packets by the sender since starting transmission on this connection. In the case of RTP, the count is not reset if the sender changes its SSRC identifier, for example as a result of a ModifyConnection command. The value is zero if the connection was always set in "receive only" mode and no signals were applied to the connection.

Number of packets received:

The total number of media packets received by the sender since starting reception on this connection. In the case of RTP, the count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was always set in "send only" mode.

Number of octets received:

The total number of payload octets (i.e., not including header, e.g., RTP, or padding) transmitted in media packets by the sender since starting transmission on this connection. In the case of RTP, the count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was always set in "send only" mode.

Number of packets lost:

The total number of media packets that have been lost since the beginning of reception. This number is defined to be the number of packets expected less the number of packets actually received, where the number of packets received includes any which are late or duplicates. For RTP, the count includes packets received from different SSRC, if the sender used several values. Thus packets that arrive late are not counted as lost, and the loss may be negative if there are duplicates. The count includes packets received from different SSRC, if the sender used several values. The number of packets expected is defined to be the extended last sequence number received, as defined next, less the initial sequence number received. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was always set in "send only" mode.

Interarrival jitter:

An estimate of the statistical variance of the media packet interarrival time measured in milliseconds and expressed as an unsigned integer. For RTP, the interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. Detailed computation algorithms are found in RFC 1889. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was always set in "send only" mode.

Average transmission delay:

An estimate of the network latency, expressed in milliseconds. For RTP, this is the average value of the difference between the NTP timestamp indicated by the senders of the RTCP messages and the NTP timestamp of the receivers, measured when the messages are received. The average is obtained by summing all the estimates,

then dividing by the number of RTCP messages that have been received. When the gateway's clock is not synchronized by NTP, the latency value can be computed as one half of the round trip delay, as measured through RTCP. When the gateway cannot compute the one way delay or the round trip delay, the parameter conveys a null value.

For a detailed definition of these variables, refer to RFC 1889.

When the connection was set up over a LOCAL interconnect, the meaning of these parameters is defined as follows:

Number of packets sent:
Not significant - MAY be omitted.

Number of octets sent:
The total number of payload octets transmitted over the local connection.

Number of packets received:
Not significant - MAY be omitted.

Number of octets received:
The total number of payload octets received over the connection.

Number of packets lost:
Not significant - MAY be omitted. A value of zero is assumed.

Interarrival jitter:
Not significant - MAY be omitted. A value of zero is assumed.

Average transmission delay:
Not significant - MAY be omitted. A value of zero is assumed.

The set of connection parameters can be extended. Also, the meaning may be further defined by other types of networks which MAY furthermore elect to not return all, or even any, of the above specified parameters.

The command may optionally contain an encapsulated Notification Request command, in which case a RequestIdentifier parameter MUST be present, as well as, optionally, other parameters of the NotificationRequest with the exception of the EndpointId, which is not replicated. The encapsulated NotificationRequest is executed simultaneously with the deletion of the connection. For example, when a user hang-up is notified, the gateway should be instructed to delete the connection and to start looking for an off-hook event.

This can be accomplished in a single DeleteConnection command, by also transmitting the RequestedEvents parameters, for the off-hook event, and an empty SignalRequests parameter.

When these parameters are present, the DeleteConnection and the NotificationRequest must be synchronized, which means that both MUST be accepted, or both MUST be refused.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are included with the normal parameters of the DeleteConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the DeleteConnection command. If the DeleteConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.8 DeleteConnection (from the gateway)

In some rare circumstances, a gateway may have to clear a connection, for example because it has lost the resource associated with the connection, or because it has detected that the endpoint no longer is capable or willing to send or receive media. The gateway may then terminate the connection by using a variant of the DeleteConnection command:

```
      ReturnCode,
      [PackageList]
      <-- DeleteConnection(CallId,
                           EndpointId,
                           ConnectionId,
                           ReasonCode,
                           Connection-parameters)
```

The EndpointId, in this form of the DeleteConnection command, MUST be fully qualified. Wildcard conventions MUST NOT be used.

The ReasonCode is a text string starting with a numeric reason code and optionally followed by a descriptive text string. The reason code indicates the cause of the DeleteConnection. A list of reason codes can be found in Section 2.5.

In addition to the call, endpoint and connection identifiers, the gateway will also send the connection parameters that would have been returned to the Call Agent in response to a DeleteConnection command.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

Note that use of this command is generally discouraged and should only be done as a last resort. If a connection can be sustained, deletion of it should be left to the discretion of the Call Agent which is in a far better position to make intelligent decisions in this area.

2.3.9 DeleteConnection (multiple connections from the Call Agent)

A variation of the DeleteConnection function can be used by the Call Agent to delete multiple connections at the same time. Note that encapsulating other commands with this variation of the DeleteConnection command is not permitted. The command can be used to delete all connections that relate to a Call for an endpoint:

```
ReturnCode,
[PackageList]
<-- DeleteConnection(CallId,
                     EndpointId)
```

The EndpointId, in this form of the DeleteConnection command, MUST NOT use the "any of" wildcard. All connections for the endpoint(s) with the CallId specified will be deleted. Note that the command will still succeed if there were no connections with the CallId specified, as long as the EndpointId was valid. However, if the EndpointId is invalid, the command will fail. The command does not return any individual statistics or call parameters.

It can also be used to delete all connections that terminate in a given endpoint:

```
ReturnCode,
[PackageList]
<-- DeleteConnection(EndpointId)
```

The EndpointId, in this form of the DeleteConnection command, MUST NOT use the "any of" wildcard. Again, the command succeeds even if there were no connections on the endpoint(s).

Finally, Call Agents can take advantage of the hierarchical structure of endpoint names to delete all the connections that belong to a group of endpoints. In this case, the "local name" component of the EndpointId will be specified using the "all of" wildcarding convention. The "any of" convention SHALL NOT be used. For example, if endpoint names are structured as the combination of a physical interface name and a circuit number, as in "X35V3+A4/13", the Call Agent may replace the circuit number by the "all of" wild card character "*", as in "X35V3+A4/*". This "wildcard" command instructs the gateway to delete all the connections that were attached to circuits connected to the physical interface "X35V3+A4".

After all the connections have been deleted, any loopback that has been requested for the connections MUST be cancelled by the gateway.

This command does not return any individual statistics or call parameters.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.10 AuditEndpoint

The AuditEndPoint command can be used by the Call Agent to find out the status of a given endpoint.

```
      ReturnCode,
      EndPointIdList,|{
      [RequestedEvents,]
      [QuarantineHandling,]
      [DigitMap,]
      [SignalRequests,]
      [RequestIdentifier,]
      [NotifiedEntity,]
      [ConnectionIdentifiers,]
      [DetectEvents,]
      [ObservedEvents,]
      [EventStates,]
      [BearerInformation,]
      [RestartMethod,]
      [RestartDelay,]
      [ReasonCode,]
      [MaxMGCPDatagram,]
      [Capabilities]}
      [PackageList]
      <-- AuditEndPoint(EndpointId,
                        [RequestedInfo])
```

The EndpointId identifies the endpoint(s) being audited. The "any of" wildcard convention MUST NOT be used.

The EndpointId identifies the endpoint(s) being audited. The "all of" wildcard convention can be used to start auditing of a group of endpoints (regardless of their service-state). If this convention is used, the gateway SHALL return the list of endpoint identifiers that match the wildcard in the EndPointIdList parameter, which is simply one or more SpecificEndpointIds (each supplied separately). In the case where the "all of" wildcard is used, RequestedInfo SHOULD NOT be included (if it is included, it MUST be ignored). Note that the use of the "all of" wildcard can potentially generate a large EndPointIdList. If the resulting EndPointIdList is considered too large, the gateway returns an error (error code 533 - response too large, is RECOMMENDED).

When a non-wildcard EndpointId is specified, the (possibly empty) RequestedInfo parameter describes the information that is requested for the EndpointId specified. The following endpoint info can be audited with this command:

RequestedEvents, DigitMap, SignalRequests, RequestIdentifier, QuarantineHandling, NotifiedEntity, ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, BearerInformation, RestartMethod, RestartDelay, ReasonCode, PackageList, MaxMGCPDatagram, and Capabilities.

The list may be extended by extension parameters. The response will in turn include information about each of the items for which auditing info was requested. Supported parameters with empty values MUST always be returned. However, if an endpoint is queried about a parameter it does not understand, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response:

* RequestedEvents: The current value of RequestedEvents the endpoint is using including the action(s) and event parameters associated with each event - if no actions are included, the default action is assumed. Persistent events are included in the list. If an embedded NotificationRequest is active, the RequestedEvents will reflect the events requested in the embedded NotificationRequest, not any surrounding RequestedEvents (whether embedded or not).

* DigitMap: The digit map the endpoint is currently using. The parameter will be empty if the endpoint does not have a digit map.

* SignalRequests: A list of the; Time-Out signals that are currently active, On/Off signals that are currently "on" for the endpoint (with or without parameter), and any pending Brief signals. Time-Out signals that have timed-out, and currently playing Brief signals are not included. Any signal parameters included in the original SignalRequests will be included.

* RequestIdentifier: The RequestIdentifier for the last NotificationRequest received by this endpoint (includes NotificationRequests encapsulated in other commands). If no NotificationRequest has been received since reboot/restart, the value zero will be returned.

* QuarantineHandling: The QuarantineHandling for the last NotificationRequest received by this endpoint. If QuarantineHandling was not included, or no notification request has been received, the default values will be returned.

* DetectEvents: The value of the most recently received DetectEvents parameter plus any persistent events implemented by the endpoint. If no DetectEvents parameter has been received, the (possibly empty) list only includes persistent events.

* NotifiedEntity: The current "notified entity" for the endpoint.

* ConnectionIdentifiers: The list of ConnectionIdentifiers for all connections that currently exist for the specified endpoint.

* ObservedEvents: The current list of observed events for the endpoint.

* EventStates: For events that have auditable states associated with them, the event corresponding to the state the endpoint is in, e.g., off-hook if the endpoint is off-hook. Note that the definition of the individual events will state if the event in question has an auditable state associated with it.

* BearerInformation: The value of the last received BearerInformation parameter for this endpoint (this includes the case where BearerInformation was provisioned). The parameter will be empty if the endpoint has not received a BearerInformation parameter and a value was also not provisioned.

* RestartMethod: "restart" if the endpoint is in-service and operation is normal, or if the endpoint is in the process of becoming in-service (a non-zero RestartDelay will indicate the latter). Otherwise, the value of the restart method parameter in the last RestartInProgress command issued (or should have been issued) by the endpoint. Note that a "disconnected" endpoint will thus only report "disconnected" as long as it actually is disconnected, and "restart" will be reported once it is no longer disconnected. Similarly, "cancel-graceful" will not be reported, but "graceful" might (see Section 4.4.5 for further details).

* RestartDelay: The value of the restart delay parameter if a RestartInProgress command was to be issued by the endpoint at the time of this response, or zero if the command would not include this parameter.

* ReasonCode: The value of the ReasonCode parameter in the last RestartInProgress or DeleteConnection command issued by the gateway for the endpoint, or the special value 000 if the endpoint's state is normal.

* PackageList: The packages supported by the endpoint including package version numbers. For backwards compatibility, support for the parameter is OPTIONAL although implementations with package versions higher than zero SHOULD support it.

* MaxMGCPDatagram: The maximum size of an MGCP datagram in bytes that can be received by the endpoint (see Section 3.5.4). The value excludes any lower layer overhead. For backwards compatibility, support for this parameter is OPTIONAL. The default maximum MGCP datagram size SHOULD be assumed if a value is not returned.

* Capabilities: The capabilities for the endpoint similar to the LocalConnectionOptions parameter and including packages and connection modes. Extensions MAY be included as well. If any unknown capabilities are reported, they MUST simply be ignored. If there is a need to specify that some parameters, such as e.g., silence suppression, are only compatible with some codecs, then the gateway MUST return several capability sets, each of which may include:

  - Compression Algorithm: A list of supported codecs. The rest of the parameters in the capability set will apply to all codecs specified in this list.

  - Packetization Period: A single value or a range may be specified.

  - Bandwidth: A single value or a range corresponding to the range for packetization periods may be specified (assuming no silence suppression).

  - Echo Cancellation: Whether echo cancellation is supported or not for the endpoint.

  - Silence Suppression: Whether silence suppression is supported or not.

  - Gain Control: Whether gain control is supported or not.

  - Type of Service: Whether type of service is supported or not.

  - Resource Reservation: Whether resource reservation is supported or not.

  - Security: Whether media encryption is supported or not.

  - Type of network: The type(s) of network supported.

  - Packages: A list of packages supported. The first package in the list will be the default package.

  - Modes: A list of supported connection modes.

The Call Agent may then decide to use the AuditConnection command to obtain further information about the connections.

If no info was requested and the EndpointId refers to a valid endpoint (in-service or not), the gateway simply returns a positive acknowledgement.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

Note that PackageList MAY also be included with error code 518 (unsupported package).

2.3.11 AuditConnection

The AuditConnection command can be used by the Call Agent to retrieve the parameters attached to a connection.

```
      ReturnCode,
      [CallId,]
      [NotifiedEntity,]
      [LocalConnectionOptions,]
      [Mode,]
      [RemoteConnectionDescriptor,]
      [LocalConnectionDescriptor,]
      [ConnectionParameters,]
      [PackageList]
      <-- AuditConnection(EndpointId,
                          ConnectionId,
                          RequestedInfo)
```

The EndpointId parameter specifies the endpoint that handles the connection. The wildcard conventions SHALL NOT be used.

The ConnectionId parameter is the identifier of the audited connection, within the context of the specified endpoint.

The (possibly empty) RequestedInfo describes the information that is requested for the ConnectionId within the EndpointId specified. The following connection info can be audited with this command:

```
   CallId, NotifiedEntity, LocalConnectionOptions, Mode,
   RemoteConnectionDescriptor, LocalConnectionDescriptor,
   ConnectionParameters
```

The AuditConnection response will in turn include information about each of the items auditing info was requested for:

* CallId, the CallId for the call the connection belongs to.

* NotifiedEntity, the current "notified entity" for the Connection. Note this is the same as the "notified entity" for the endpoint (included here for backwards compatibility).

* LocalConnectionOptions, the most recent LocalConnectionOptions parameters that was actually supplied for the connection (omitting LocalConnectionOptions from a command thus does not change this value). Note that default parameters omitted from the most recent LocalConnectionOptions will not be included. LocalConnectionOptions that retain their value across ModifyConnection commands and which have been included in a previous command for the connection are also included, regardless of whether they were supplied in the most recent LocalConnectionOptions or not.

* Mode, the current mode of the connection.

* RemoteConnectionDescriptor, the RemoteConnectionDescriptor that was supplied to the gateway for the connection.

* LocalConnectionDescriptor, the LocalConnectionDescriptor the gateway supplied for the connection.

* ConnectionParameters, the current values of the connection parameters for the connection.

If no info was requested and the EndpointId is valid, the gateway simply checks that the connection exists, and if so returns a positive acknowledgement. Note, that by definition, the endpoint must be in-service for this to happen, as out-of-service endpoints do not have any connections.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.3.12 RestartInProgress

The RestartInProgress command is used by the gateway to signal that an endpoint, or a group of endpoints, is put in-service or out-of-service.

```
      ReturnCode,
      [NotifiedEntity,]
      [PackageList]
      <-- RestartInProgress(EndPointId,
                            RestartMethod,
                            [RestartDelay,]
                            [ReasonCode])
```

The EndPointId identifies the endpoint(s) that are put in-service or out-of-service. The "all of" wildcard convention may be used to apply the command to a group of endpoints managed by the same Call Agent, such as for example all endpoints that are attached to a specified interface, or even all endpoints that are attached to a given gateway. The "any of" wildcard convention SHALL NOT be used.

The RestartMethod parameter specifies the type of restart. The following values have been defined:

* A "graceful" restart method indicates that the specified endpoints will be taken out-of-service after the specified delay. The established connections are not yet affected, but the Call Agent SHOULD refrain from establishing new connections, and SHOULD try to gracefully tear down the existing connections.

* A "forced" restart method indicates that the specified endpoints are taken abruptly out-of-service. The established connections, if any, are lost.

* A "restart" method indicates that service will be restored on the endpoints after the specified "restart delay", i.e., the endpoints will be in-service. The endpoints are in their clean default state and there are no connections that are currently established on the endpoints.

* A "disconnected" method indicates that the endpoint has become disconnected and is now trying to establish connectivity (see Section 4.4.7). The "restart delay" specifies the number of seconds the endpoint has been disconnected. Established connections are not affected.

* A "cancel-graceful" method indicates that a gateway is canceling a previously issued "graceful" restart command. The endpoints are still in-service.

The list of restart methods may be extended.

The optional "restart delay" parameter is expressed as a number of seconds. If the number is absent, the delay value MUST be considered null (i.e., zero). In the case of the "graceful" method, a null delay indicates that the Call Agent SHOULD simply wait for the natural termination of the existing connections, without establishing new connections. The restart delay is always considered null in the case of the "forced" and "cancel-graceful" methods, and hence the "restart delay" parameter MUST NOT be used with these restart methods. When the gateway sends a "restart" or "graceful"

RestartInProgress message with a non-zero restart delay, the gateway SHOULD send an updated RestartInProgress message after the "restart delay" has passed.

A restart delay of null for the "restart" method indicates that service has already been restored. This typically will occur after gateway startup/reboot. To mitigate the effects of a gateway IP address change as a result of a re-boot, the Call Agent MAY wish to either flush its DNS cache for the gateway's domain name or resolve the gateway's domain name by querying the DNS regardless of the TTL of a current DNS resource record for the restarted gateway.

The optional reason code parameter indicates the cause of the restart.

Gateways SHOULD send a "graceful" or "forced" RestartInProgress message (for the relevant endpoints) as a courtesy to the Call Agent when they are taken out-of-service, e.g., by being shutdown, or taken out-of-service by a network management system, however the Call Agent cannot rely on always receiving such a message. Gateways MUST send a "restart" RestartInProgress message (for the relevant endpoints) with a null delay to their Call Agent when they are back in-service according to the restart procedure specified in Section 4.4.6 - Call Agents can rely on receiving this message. Also, gateways MUST send a "disconnected" RestartInProgress message (for the relevant endpoints) to their current "notified entity" according to the "disconnected" procedure specified in Section 4.4.7.

The RestartInProgress message will be sent to the current "notified entity" for the EndpointId in question. It is expected that a default Call Agent, i.e., "notified entity", has been provisioned so that after a reboot/restart, the default Call Agent will always be the "notified entity" for the endpoint. Gateways SHOULD take full advantage of wild-carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

A NotifiedEntity may additionally be returned with the response to the RestartInProgress from the Call Agent - this SHOULD normally only be done in response to "restart" or "disconnected" (see also Section 4.4.6 and 4.4.7):

* If the response indicated success (return code 200 - transaction executed), the restart in question completed successfully, and the NotifiedEntity returned is the new "notified entity" for the endpoint(s).

* If the response from the Call Agent indicated an error, the restart in question did not complete successfully. If a NotifiedEntity parameter was included in the response returned, it specifies a new "notified entity" for the endpoint(s), which MUST be used when retrying the restart in question (as a new transaction). This SHOULD only be done with error code 521 (endpoint redirected).

Note that the above behavior for returning a NotifiedEntity in the response is only defined for RestartInProgress responses and SHOULD NOT be done for responses to other commands. Any other behavior is undefined.

PackageList is a list of supported packages that MAY be included with error code 518 (unsupported package).

2.4 Return Codes and Error Codes

All MGCP commands are acknowledged. The acknowledgment carries a return code, which indicates the status of the command. The return code is an integer number, for which the following ranges of values have been defined:

* values between 000 and 099 indicate a response acknowledgement

* values between 100 and 199 indicate a provisional response

* values between 200 and 299 indicate a successful completion

* values between 400 and 499 indicate a transient error

* values between 500 and 599 indicate a permanent error

* values between 800 and 899 are package specific response codes.

A broad description of transient errors (4XX error codes) versus permanent errors (5XX error codes) is as follows:

* If a Call Agent receives a transient error, there is the expectation of the possibility that a future similar request will be honored by the endpoint. In some cases, this may require some state change in the environment of the endpoint (e.g., hook state as in the case of error codes 401 or 402; resource availability as in the case of error code 403, or bandwidth availability as in the case of error code 404).

* Permanent errors (error codes 500 to 599) indicate one or more permanent conditions either due to protocol error or incompatibility between the endpoint and the Call Agent, or because of some error condition over which the Call Agent has no control. Examples are protocol errors, requests for endpoint capabilities that do not exist, errors on interfaces associated with the endpoint, missing or incorrect information in the request or any number of other conditions which will simply not disappear with time.

The values that have been already defined are the following:

000 Response Acknowledgement.

100 The transaction is currently being executed. An actual completion message will follow later.

101 The transaction has been queued for execution. An actual completion message will follow later.

200 The requested transaction was executed normally. This return code can be used for a successful response to any command.

250 The connection was deleted. This return code can only be used for a successful response to a DeleteConnection command.

400 The transaction could not be executed, due to some unspecified transient error.

401 The phone is already off hook.

402 The phone is already on hook.

403 The transaction could not be executed, because the endpoint does not have sufficient resources at this time.

404 Insufficient bandwidth at this time.

405 The transaction could not be executed, because the endpoint is "restarting".

406 Transaction time-out. The transaction did not complete in a reasonable period of time and has been aborted.

407 Transaction aborted. The transaction was aborted by some external action, e.g., a ModifyConnection command aborted by a DeleteConnection command.

409 The transaction could not be executed because of internal overload.

410 No endpoint available. A valid "any of" wildcard was used, however there was no endpoint available to satisfy the request.

500 The transaction could not be executed, because the endpoint is unknown.

501 The transaction could not be executed, because the endpoint is not ready. This includes the case where the endpoint is out-of-service.

502 The transaction could not be executed, because the endpoint does not have sufficient resources (permanent condition).

503 "All of" wildcard too complicated.

504 Unknown or unsupported command.

505 Unsupported RemoteConnectionDescriptor. This SHOULD be used when one or more mandatory parameters or values in the RemoteConnectionDescriptor is not supported.

506 Unable to satisfy both LocalConnectionOptions and RemoteConnectionDescriptor. This SHOULD be used when the LocalConnectionOptions and RemoteConnectionDescriptor contain one or more mandatory parameters or values that conflict with each other and/or cannot be supported at the same time (except for codec negotiation failure - see error code 534).

507 Unsupported functionality. Some unspecified functionality required to carry out the command is not supported. Note that several other error codes have been defined for specific areas of unsupported functionality (e.g. 508, 511, etc.), and this error code SHOULD only be used if there is no other more specific error code for the unsupported functionality.

508 Unknown or unsupported quarantine handling.

509 Error in RemoteConnectionDescriptor. This SHOULD be used when there is a syntax or semantic error in the RemoteConnectionDescriptor.

510 The transaction could not be executed, because some unspecified protocol error was detected. Automatic recovery from such an error will be very difficult, and hence this code SHOULD only be used as a last resort.

511 The transaction could not be executed, because the command contained an unrecognized extension. This code SHOULD be used for unsupported critical parameter extensions ("X+").

512 The transaction could not be executed, because the gateway is not equipped to detect one of the requested events.

513 The transaction could not be executed, because the gateway is not equipped to generate one of the requested signals.

514 The transaction could not be executed, because the gateway cannot send the specified announcement.

515 The transaction refers to an incorrect connection-id (may have been already deleted).

516 The transaction refers to an unknown call-id, or the call-id supplied is incorrect (e.g., connection-id not associated with this call-id).

517 Unsupported or invalid mode.

518 Unsupported or unknown package. It is RECOMMENDED to include a PackageList parameter with the list of supported packages in the response, especially if the response is generated by the Call Agent.

519 Endpoint does not have a digit map.

520 The transaction could not be executed, because the endpoint is "restarting". In most cases this would be a transient error, in which case, error code 405 SHOULD be used instead. The error code is only included here for backwards compatibility.

521 Endpoint redirected to another Call Agent. The associated redirection behavior is only well-defined when this response is issued for a RestartInProgress command.

522 No such event or signal. The request referred to an event or signal that is not defined in the relevant package (which could be the default package).

523 Unknown action or illegal combination of actions.

524 Internal inconsistency in LocalConnectionOptions.

525 Unknown extension in LocalConnectionOptions. This code SHOULD be used for unsupported mandatory vendor extensions ("x+").

526 Insufficient bandwidth. In cases where this is a transient error, error code 404 SHOULD be used instead.

527 Missing RemoteConnectionDescriptor.

528 Incompatible protocol version.

529 Internal hardware failure.

530 CAS signaling protocol error.

531 Failure of a grouping of trunks (e.g., facility failure).

532 Unsupported value(s) in LocalConnectionOptions.

533 Response too large.

534 Codec negotiation failure.

535 Packetization period not supported.

536 Unknown or unsupported RestartMethod.

537 Unknown or unsupported digit map extension.

538 Event/signal parameter error (e.g., missing, erroneous, unsupported, unknown, etc.).

539 Invalid or unsupported command parameter. This code SHOULD only be used when the parameter is neither a package or vendor extension parameter.

540 Per endpoint connection limit exceeded.

541 Invalid or unsupported LocalConnectionOptions. This code SHOULD only be used when the LocalConnectionOptions is neither a package nor a vendor extension LocalConnectionOptions.

The set of return codes may be extended in a future version of the protocol. Implementations that receive an unknown or unsupported return code SHOULD treat the return code as follows:

* Unknown 0xx code treated as 000.

* Unknown 1xx code treated as 100.

* Unknown 2xx code treated as 200.

* Unknown 3xx code treated as 521.

* Unknown 4xx code treated as 400.

* Unknown 5xx-9xx code treated as 510.

2.5 Reason Codes

Reason codes are used by the gateway when deleting a connection to inform the Call Agent about the reason for deleting the connection. They may also be used in a RestartInProgress command to inform the Call Agent of the reason for the RestartInProgress.

The reason code is an integer number, and the following values have been defined:

000 Endpoint state is normal (this code is only used in response to audit requests).

900 Endpoint malfunctioning.

901 Endpoint taken out-of-service.

902 Loss of lower layer connectivity (e.g., downstream sync).

903 QoS resource reservation was lost.

904 Manual intervention.

905 Facility failure (e.g., DS-0 failure).

The set of reason codes can be extended.

2.6 Use of Local Connection Options and Connection Descriptors

As indicated previously, the normal sequence in setting up a bi-directional connection involves at least 3 steps:

1) The Call Agent asks the first gateway to "create a connection" on an endpoint. The gateway allocates resources to that connection, and responds to the command by providing a "session description" (referred to as its LocalConnectionDescriptor). The session description contains the information necessary for another party to send packets towards the newly created connection.

2) The Call Agent then asks the second gateway to "create a connection" on an endpoint. The command carries the "session description" provided by the first gateway (now referred to as the RemoteConnectionDescriptor). The gateway allocates resources to that connection, and responds to the command by providing its own "session description" (LocalConnectionDescriptor).

3) The Call Agent uses a "modify connection" command to provide this second "session description" (now referred to as the RemoteConnectionDescriptor) to the first endpoint. Once this is done, communication can proceed in both directions.

When the Call Agent issues a Create or Modify Connection command, there are thus three parameters that determine the media supported by that connection:

* LocalConnectionOptions: Supplied by the Call Agent to control the media parameters used by the gateway for the connection. When supplied, the gateway MUST conform to these media parameters until either the connection is deleted, or a ModifyConnection command with new media parameters (LocalConnectionOptions or RemoteConnectionDescriptor) is received.

* RemoteConnectionDescriptor: Supplied by the Call Agent to convey the media parameters supported by the other side of the connection. When supplied, the gateway MUST conform to these media parameters until either the connection is deleted, or a ModifyConnection command with new media parameters (LocalConnectionOptions or RemoteConnectionDescriptor) is received.

* LocalConnectionDescriptor: Supplied by the gateway to the Call Agent to convey the media parameters it supports for the connection. When supplied, the gateway MUST honor the media parameters until either the connection is deleted, or the gateway issues a new LocalConnectionDescriptor for that connection.

In determining which codec(s) to provide in the LocalConnectionDescriptor, there are three lists of codecs that a gateway needs to consider:

* A list of codecs allowed by the LocalConnectionOptions in the current command (either explicitly by encoding method or implicitly by bandwidth and/or packetization period).

* A list of codecs in the RemoteConnectionDescriptor in the current command.

* An internal list of codecs that the gateway can support for the connection. A gateway MAY support one or more codecs for a given connection.

Codec selection (including all relevant media parameters) can then be described by the following steps:

1. An approved list of codecs is formed by taking the intersection of the internal list of codecs and codecs allowed by the LocalConnectionOptions. If LocalConnectionOptions were not provided in the current command, the approved list of codecs thus contains the internal list of codecs.

2. If the approved list of codecs is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 - codec negotiation failure, is RECOMMENDED).

3. Otherwise, a negotiated list of codecs is formed by taking the intersection of the approved list of codecs and codecs allowed by the RemoteConnectionDescriptor. If a RemoteConnectionDescriptor was not provided in the current command, the negotiated list of codecs thus contains the approved list of codecs.

4. If the negotiated list of codecs is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 - codec negotiation failure, is RECOMMENDED).

5. Otherwise, codec negotiation has succeeded, and the negotiated list of codecs is returned in the LocalConnectionDescriptor.

Note that both LocalConnectionOptions and the RemoteConnectionDescriptor can contain a list of codecs ordered by preference. When both are supplied in the current command, the gateway MUST adhere to the preferences provided in the LocalConnectionOptions.

2.7 Resource Reservations

The gateways can be instructed to perform a reservation, for example using RSVP, on a given connection. When a reservation is needed, the call agent will specify the reservation profile to be used, which is either "controlled load" or "guaranteed service". The absence of reservation can be indicated by asking for the "best effort" service, which is the default value of this parameter in a CreateConnection command. For a ModifyConnection command, the default is simply to retain the current value. When reservation has been asked on a connection, the gateway will:

* start emitting RSVP "PATH" messages if the connection is in "send-only", "send-receive", "conference", "network loop back" or "network continuity test" mode (if a suitable remote connection descriptor has been received,).

* start emitting RSVP "RESV" messages as soon as it receives "PATH" messages if the connection is in "receive-only", "send-receive", "conference", "network loop back" or "network continuity test" mode.

The RSVP filters will be deduced from the characteristics of the connection. The RSVP resource profiles will be deduced from the connection's codecs, bandwidth and packetization period.

3. Media Gateway Control Protocol

The Media Gateway Control Protocol (MGCP) implements the media gateway control interface as a set of transactions. The transactions are composed of a command and a mandatory response. There are nine commands:

* EndpointConfiguration

* CreateConnection

* ModifyConnection

* DeleteConnection

* NotificationRequest

* Notify

* AuditEndpoint

* AuditConnection

* RestartInProgress

The first five commands are sent by the Call Agent to a gateway. The Notify command is sent by the gateway to the Call Agent. The gateway may also send a DeleteConnection as defined in Section 2.3.8. The Call Agent may send either of the Audit commands to the gateway, and the gateway may send a RestartInProgress command to the Call Agent.

3.1 General Description

All commands are composed of a Command header, optionally followed by a session description.

All responses are composed of a Response header, optionally followed by session description information.

Headers and session descriptions are encoded as a set of text lines, separated by a carriage return and line feed character (or, optionally, a single line-feed character). The session descriptions are preceded by an empty line.

MGCP uses a transaction identifier to correlate commands and responses. The transaction identifier is encoded as a component of the command header and repeated as a component of the response header (see sections 3.2.1.2 and 3.3).

Note that an ABNF grammar for MGCP is provided in Appendix A. Commands and responses SHALL be encoded in accordance with the grammar, which, per RFC 2234, is case-insensitive except for the SDP part. Similarly, implementations SHALL be capable of decoding commands and responses that follow the grammar. Additionally, it is RECOMMENDED that implementations tolerate additional linear white space.

Some productions allow for use of quoted strings, which can be necessary to avoid syntax problems. Where the quoted string form is used, the contents will be UTF-8 encoded [20], and the actual value provided is the unquoted string (UTF-8 encoded). Where both a quoted and unquoted string form is allowed, either form can be used provided it does not otherwise violate the grammar.

In the following, we provide additional detail on the format of MGCP commands and responses.

3.2 Command Header

The command header is composed of:

* A command line, identifying the requested action or verb, the transaction identifier, the endpoint towards which the action is requested, and the MGCP protocol version,

* A set of zero or more parameter lines, composed of a parameter name followed by a parameter value.

Unless otherwise noted or dictated by other referenced standards (e.g., SDP), each component in the command header is case insensitive. This goes for verbs as well as parameters and values, and hence all comparisons MUST treat upper and lower case as well as combinations of these as being equal.

3.2.1 Command Line

The command line is composed of:

* The name of the requested verb,

* The identification of the transaction,

* The name of the endpoint(s) that are to execute the command (in notifications or restarts, the name of the endpoint(s) that is issuing the command),

* The protocol version.

  These four items are encoded as strings of printable ASCII characters, separated by white spaces, i.e., the ASCII space (0x20) or tabulation (0x09) characters. It is RECOMMENDED to use exactly one ASCII space separator. However, MGCP entities MUST be able to parse messages with additional white space characters.

3.2.1.1 Coding of the Requested Verb

The verbs that can be requested are encoded as four letter upper or lower case ASCII codes (comparisons SHALL be case insensitive) as defined in the following table:

Verb	Code
EndpointConfiguration	EPCF
CreateConnection	CRCX
ModifyConnection	MDCX
DeleteConnection	DLCX
NotificationRequest	RQNT
Notify	NTFY
AuditEndpoint	AUEP
AuditConnection	AUCX
RestartInProgress	RSIP

The transaction identifier is encoded as a string of up to 9 decimal digits. In the command line, it immediately follows the coding of the verb.

New verbs may be defined in further versions of the protocol. It may be necessary, for experimentation purposes, to use new verbs before they are sanctioned in a published version of this protocol. Experimental verbs MUST be identified by a four letter code starting with the letter X, such as for example XPER.

3.2.1.2 Transaction Identifiers

MGCP uses a transaction identifier to correlate commands and responses. A gateway supports two separate transaction identifier name spaces:

* a transaction identifier name space for sending transactions, and

* a transaction identifier name space for receiving transactions.

At a minimum, transaction identifiers for commands sent to a given gateway MUST be unique for the maximum lifetime of the transactions within the collection of Call Agents that control that gateway. Thus, regardless of the sending Call Agent, gateways can always detect duplicate transactions by simply examining the transaction identifier. The coordination of these transaction identifiers between Call Agents is outside the scope of this specification though.

Transaction identifiers for all commands sent from a given gateway MUST be unique for the maximum lifetime of the transactions regardless of which Call Agent the command is sent to. Thus, a Call Agent can always detect a duplicate transaction from a gateway by the combination of the domain-name of the endpoint and the transaction identifier.

The transaction identifier is encoded as a string of up to nine decimal digits. In the command lines, it immediately follows the coding of the verb.

Transaction identifiers have values between 1 and 999,999,999 (both included). Transaction identifiers SHOULD NOT use any leading zeroes, although equality is based on numerical value, i.e., leading zeroes are ignored. An MGCP entity MUST NOT reuse a transaction identifier more quickly than three minutes after completion of the previous command in which the identifier was used.

3.2.1.3 Coding of the Endpoint Identifiers and Entity Names

The endpoint identifiers and entity names are encoded as case insensitive e-mail addresses, as defined in RFC 821, although with some syntactic restrictions on the local part of the name. Furthermore, both the local endpoint name part and the domain name part can each be up to 255 characters. In these addresses, the domain name identifies the system where the endpoint is attached, while the left side identifies a specific endpoint or entity on that system.

Examples of such addresses are:

hrd4/56@gw23.example.net	Circuit number 56 in interface "hrd4" of the Gateway 23 of the "Example" network
Call-agent@ca.example.net	Call Agent for the "example" network
Busy-signal@ann12.example.net	The "busy signal" virtual endpoint in the announcement server number 12.

The name of a notified entity is expressed with the same syntax, with the possible addition of a port number as in:

```
Call-agent@ca.example.net:5234
```

In case the port number is omitted from the notified entity, the default MGCP Call Agent port (2727) MUST be used.

3.2.1.4 Coding of the Protocol Version

The protocol version is coded as the keyword MGCP followed by a white space and the version number, and optionally followed by a profile name. The version number is composed of a major version, coded by a decimal number, a dot, and a minor version number, coded as a decimal number. The version described in this document is version 1.0.

The profile name, if present, is represented by white-space separated strings of visible (printable) characters extending to the end of the line. Profile names may be defined for user communities who want to apply restrictions or other profiling to MGCP.

In the initial messages, the version will be coded as:

MGCP 1.0

An entity that receives a command with a protocol version it does not support, MUST respond with an error (error code 528 - incompatible protocol version, is RECOMMENDED). Note that this applies to unsupported profiles as well.

3.2.2 Parameter Lines

Parameter lines are composed of a parameter name, which in most cases is composed of one or two characters, followed by a colon, optional white space(s) and the parameter value. The parameters that can be present in commands are defined in the following table:

Parameter name	Code	Parameter value
BearerInformation	B	See description (3.2.2.1).
CallId	C	See description (3.2.2.2).
Capabilities	A	See description (3.2.2.3).
ConnectionId	I	See description (3.2.2.5).
ConnectionMode	M	See description (3.2.2.6).
ConnectionParameters	P	See description (3.2.2.7).
DetectEvents	T	See description (3.2.2.8).
DigitMap	D	A text encoding of a digit map.
EventStates	ES	See description (3.2.2.9).
LocalConnectionOptions	L	See description (3.2.2.10).
MaxMGCPDatagram	MD	See description (3.2.2.11).
NotifiedEntity	N	An identifier, in RFC 821 format, composed of an arbitrary string and of the domain name of the requesting entity, possibly completed by a port number, as in: Call-agent@ca.example.net:5234 See also Section 3.2.1.3.
ObservedEvents	O	See description (3.2.2.12).
PackageList	PL	See description (3.2.2.13).
QuarantineHandling	Q	See description (3.2.2.14).
ReasonCode	E	A string with a 3 digit integer optionally followed by a set of arbitrary characters (3.2.2.15).
RequestedEvents	R	See description (3.2.2.16).
RequestedInfo	F	See description (3.2.2.17).
RequestIdentifier	X	See description (3.2.2.18).
ResponseAck	K	See description (3.2.2.19).
RestartDelay	RD	A number of seconds, encoded as a decimal number.
RestartMethod	RM	See description (3.2.2.20).
SecondConnectionId	I2	Connection Id.
SecondEndpointId	Z2	Endpoint Id.
SignalRequests	S	See description (3.2.2.21).
SpecificEndPointId	Z	An identifier, in RFC 821 format, composed of an arbitrary string, followed by an "@" followed by the domain name of the gateway to which this endpoint is attached. See also Section 3.2.1.3.

RemoteConnection-Descriptor	RC	Session Description.
LocalConnection-Descriptor	LC	Session Description.

The parameters are not necessarily present in all commands. The following table provides the association between parameters and commands. The letter M stands for mandatory, O for optional and F for forbidden. Unless otherwise specified, a parameter MUST NOT be present more than once.

Parameter name	EP CF	CR CX	MD CX	DL CX	RQ NT	NT FY	AU EP	AU CX	RS IP
BearerInformation	O*	O	O	O	O	F	F	F	F
CallId	F	M	M	O	F	F	F	F	F
Capabilities	F	F	F	F	F	F	F	F	F
ConnectionId	F	F	M	O	F	F	F	M	F
ConnectionMode	F	M	O	F	F	F	F	F	F
Connection-Parameters	F	F	F	O*	F	F	F	F	F
DetectEvents	F	O	O	O	O	F	F	F	F
DigitMap	F	O	O	O	O	F	F	F	F
EventStates	F	F	F	F	F	F	F	F	F
LocalConnection-Options	F	O	O	F	F	F	F	F	F
MaxMGCPDatagram	F	F	F	F	F	F	F	F	F
NotifiedEntity	F	O	O	O	O	O	F	F	F
ObservedEvents	F	F	F	F	F	M	F	F	F
PackageList	F	F	F	F	F	F	F	F	F
QuarantineHandling	F	O	O	O	O	F	F	F	F
ReasonCode	F	F	F	O	F	F	F	F	O
RequestedEvents	F	O	O	O	O*	F	F	F	F
RequestIdentifier	F	O*	O*	O*	M	M	F	F	F
RequestedInfo	F	F	F	F	F	F	O	M	F
ResponseAck	O	O	O	O	O	O	O	O	O
RestartDelay	F	F	F	F	F	F	F	F	O
RestartMethod	F	F	F	F	F	F	F	F	M
SecondConnectionId	F	F	F	F	F	F	F	F	F
SecondEndpointId	F	O	F	F	F	F	F	F	F
SignalRequests	F	O	O	O	O*	F	F	F	F
SpecificEndpointId	F	F	F	F	F	F	F	F	F
RemoteConnection-Descriptor	F	O	O	F	F	F	F	F	F
LocalConnection-Descriptor	F	F	F	F	F	F	F	F	F

Notes (*):

* The BearerInformation parameter is only conditionally optional as explained in Section 2.3.2.

* The RequestIdentifier parameter is optional in connection creation, modification and deletion commands, however it becomes REQUIRED if the command contains an encapsulated notification request.

* The RequestedEvents and SignalRequests parameters are optional in the NotificationRequest. If these parameters are omitted the corresponding lists will be considered empty.

* The ConnectionParameters parameter is only valid in a DeleteConnection request sent by the gateway.

The set of parameters can be extended in two different ways:

* Package Extension Parameters (preferred)

* Vendor Extension Parameters

Package Extension Parameters are defined in packages which provides the following benefits:

* a registration mechanism (IANA) for the package name.

* a separate name space for the parameters.

* a convenient grouping of the extensions.

* a simple way to determine support for them through auditing.

The package extension mechanism is the preferred extension method.

Vendor extension parameters can be used if implementers need to experiment with new parameters, for example when developing a new application of MGCP. Vendor extension parameters MUST be identified by names that start with the string "X-" or "X+", such as for example:

```
X-Flower: Daisy
```

Parameter names that start with "X+" are critical parameter extensions. An MGCP entity that receives a critical parameter extension that it cannot understand MUST refuse to execute the command. It SHOULD respond with error code 511 (unrecognized extension).

Parameter names that start with "X-" are non-critical parameter extensions. An MGCP entity that receives a non-critical parameter extension that it cannot understand MUST simply ignore that parameter.

Note that vendor extension parameters use an unmanaged name space, which implies a potential for name clashing. Vendors are consequently encouraged to include some vendor specific string, e.g., vendor name, in their vendor extensions.

3.2.2.1 BearerInformation

The values of the bearer information are encoded as a comma separated list of attributes, which are represented by an attribute name, and possibly followed by a colon and an attribute value.

The only attribute that is defined is the "encoding" (code "e") attribute, which MUST have one of the values "A" (A-law) or "mu" (mu-law).

An example of bearer information encoding is:

```
B: e:mu
```

The set of bearer information attributes may be extended through packages.

3.2.2.2 CallId

The Call Identifier is encoded as a hexadecimal string, at most 32 characters in length. Call Identifiers are compared as strings rather than numerical values.

3.2.2.3 Capabilities

Capabilities inform the Call Agent about endpoints' capabilities when audited. The encoding of capabilities is based on the Local Connection Options encoding for the parameters that are common to both, although a different parameter line code is used ("A"). In addition, capabilities can also contain a list of supported packages, and a list of supported modes.

The parameters used are:

A list of supported codecs.
: The following parameters will apply to all codecs specified in this list. If there is a need to specify that some parameters, such as e.g., silence suppression, are only compatible with some codecs, then the gateway will return several Capability parameters; one for each set of codecs.

Packetization Period:
: A range may be specified.

Bandwidth:
A range corresponding to the range for packetization periods may be specified (assuming no silence suppression). If absent, the values will be deduced from the codec type.

Echo Cancellation:
"on" if echo cancellation is supported, "off" otherwise. The default is support.

Silence Suppression:
"on" if silence suppression is supported for this codec, "off" otherwise. The default is support.

Gain Control:
"0" if gain control is not supported, all other values indicate support for gain control. The default is support.

Type of Service:
The value "0" indicates no support for type of service, all other values indicate support for type of service. The default is support.

Resource Reservation Service:
The parameter indicates the reservation services that are supported, in addition to best effort. The value "g" is encoded when the gateway supports both the guaranteed and the controlled load service, "cl" when only the controlled load service is supported. The default is "best effort".

Encryption Key:
Encoding any value indicates support for encryption. Default is no support which is implied by omitting the parameter.

Type of network:
The keyword "nt", followed by a colon and a semicolon separated list of supported network types. This parameter is optional.

Packages:
The packages supported by the endpoint encoded as the keyword "v", followed by a colon and a character string. If a list of values is specified, these values will be separated by a semicolon. The first value specified will be the default package for the endpoint.

Modes:
The modes supported by this endpoint encoded as the keyword "m", followed by a colon and a semicolon-separated list of supported connection modes for this endpoint.

Lack of support for a capability can also be indicated by excluding the parameter from the capability set.

An example capability is:

```
A: a:PCMU;G728, p:10-100, e:on, s:off, t:1, v:L,
                          m:sendonly;recvonly;sendrecv;inactive
```

The carriage return above is included for formatting reasons only and is not permissible in a real implementation.

If multiple capabilities are to be returned, each will be returned as a separate capability line.

Since Local Connection Options can be extended, the list of capability parameters can also be extended. Individual extensions may define how they are reported as capabilities. If no such definition is provided, the following defaults apply:

* Package Extension attributes: The individual attributes are not reported. Instead, the name of the package is simply reported in the list of supported packages.

* Vendor Extension attributes: The name of the attribute is reported without any value.

* Other Extension attributes: The name of the attribute is reported without any value.

3.2.2.4 Coding of Event Names

Event names are composed of an optional package name, separated by a slash (/) from the name of the actual event (see Section 2.1.7). The wildcard character star ("*") can be use to refer to all packages. The event name can optionally be followed by an at sign (@) and the identifier of a connection (possibly using a wildcard) on which the event should be observed. Event names are used in the RequestedEvents, SignalRequests, ObservedEvents, DetectEvents, and EventStates parameters.

Events and signals may be qualified by parameters defined for the event/signal. Such parameters may be enclosed in double-quotes (in fact, some parameters MUST be enclosed in double-quotes due to syntactic restrictions) in which case they are UTF-8 encoded [20].

The parameter name "!" (exclamation point) is reserved for future use for both events and signals.

Each signal has one of the following signal-types associated with it: On/Off (OO), Time-out (TO), or Brief (BR). (These signal types are specified in the package definitions, and are not present in the messages.) On/Off signals can be parameterized with a "+" to turn the signal on, or a "-" to turn the signal off. If an on/off signal is not parameterized, the signal is turned on. Both of the following will turn the vmwi signal (from the line package "L") on:

```
L/vmwi(+)
L/vmwi
```

In addition to "!", "+" and "-", the signal parameter "to" is reserved as well. It can be used with Time-Out signals to override the default time-out value for the current request. A decimal value in milliseconds will be supplied. The individual signal and/or package definition SHOULD indicate if this parameter is supported for one or more TO signals in the package. If not indicated, TO signals in package version zero are assumed to not support it, whereas TO signals in package versions one or higher are assumed to support it. By default, a supplied time-out value MAY be rounded to the nearest non-zero value divisible by 1000, i.e., whole second. The individual signal and/or package definition may define other rounding rules. All new package and TO signal definitions are strongly encouraged to support the "to" signal parameter.

The following example illustrates how the "to" parameter can be used to apply a signal for 6 seconds:

```
L/rg(to=6000)
L/rg(to(6000))
```

The following are examples of event names:

L/hu	on-hook transition, in the line package
F/0	digit 0 in the MF package
hf	Hook-flash, assuming that the line package is the default package for the endpoint.
G/rt@0A3F58	Ring back signal on connection "0A3F58"

In addition, the range and wildcard notation of events can be used, instead of individual names, in the RequestedEvents and DetectEvents parameters. The event code "all" is reserved and refers to all events or signals in a package. The star sign ("*") can be used to denote "all connections", and the dollar sign ("$") can be used to denote the "current" connection (see Section 2.1.7 for details).

The following are examples of such notations:

M/[0-9]	Digits 0 to 9 in the MF package.
hf	Hook-flash, assuming that the line package is a default package for the endpoint.
[0-9*#A-D]	All digits and letters in the DTMF packages (default for endpoint).
T/all	All events in the trunk package.
R/qa@*	The quality alert event on all connections.
G/rt@$	Ringback on current connection.

3.2.2.5 ConnectionId

The Connection Identifier is encoded as a hexadecimal string, at most 32 characters in length. Connection Identifiers are compared as strings rather than numerical values.

3.2.2.6 ConnectionMode

The connection mode describes the mode of operation of the connection. The possible values are:

Mode	Meaning
M: sendonly	The gateway should only send packets
M: recvonly	The gateway should only receive packets
M: sendrecv	The gateway should send and receive packets
M: confrnce	The gateway should place the connection in conference mode
M: inactive	The gateway should neither send nor receive packets
M: loopback	The gateway should place the circuit in loopback mode.
M: conttest	The gateway should place the circuit in test mode.
M: netwloop	The gateway should place the connection in network loopback mode.
M: netwtest	The gateway should place the connection in network continuity test mode.

Note that irrespective of the connection mode, signals applied to the connection will still result in packets being sent (see Section 2.3.1).

The set of connection modes can be extended through packages.

3.2.2.7 ConnectionParameters

Connection parameters are encoded as a string of type and value pairs, where the type is either a two-letter identifier of the parameter or an extension type, and the value a decimal integer. Types are separated from value by an '=' sign. Parameters are separated from each other by a comma. Connection parameter values can contain up to nine digits. If the maximum value is reached, the counter is no longer updated, i.e., it doesn't wrap or overflow.

The connection parameter types are specified in the following table:

Connection parameter name	Code	Connection parameter value
Packets sent	PS	The number of packets that were sent on the connection.
Octets sent	OS	The number of octets that were sent on the connection.
Packets received	PR	The number of packets that were received on the connection.
Octets received	OR	The number of octets that were received on the connection.
Packets lost	PL	The number of packets that were lost on the connection as deduced from gaps in the RTP sequence number.
Jitter	JI	The average inter-packet arrival jitter, in milliseconds, expressed as an integer number.
Latency	LA	Average latency, in milliseconds, expressed as an integer number.

The set of connection parameters can be extended in two different ways:

* Package Extension Parameters (preferred)

* Vendor Extension Parameters

Package Extension Connection Parameters are defined in packages which provides the following benefits:

* A registration mechanism (IANA) for the package name.

* A separate name space for the parameters.

* A convenient grouping of the extensions.

* A simple way to determine support for them through auditing.

The package extension mechanism is the preferred extension method.

Vendor extension parameters names are composed of the string "X-" followed by a two or more letters extension parameter name.

Call agents that receive unrecognized package or vendor connection parameter extensions SHALL silently ignore these parameters.

An example of connection parameter encoding is:

```
P: PS=1245, OS=62345, PR=0, OR=0, PL=0, JI=0, LA=48
```

3.2.2.8 DetectEvents

The DetectEvents parameter is encoded as a comma separated list of events (see Section 3.2.2.4), such as for example:

```
T: L/hu,L/hd,L/hf,D/[0-9#*]
```

It should be noted, that no actions can be associated with the events, however event parameters may be provided.

3.2.2.9 EventStates

The EventStates parameter is encoded as a comma separated list of events (see Section 3.2.2.4), such as for example:

```
ES: L/hu
```

It should be noted, that no actions can be associated with the events, however event parameters may be provided.

3.2.2.10 LocalConnectionOptions

The local connection options describe the operational parameters that the Call Agent provides to the gateway in connection handling commands. These include:

* The allowed codec(s), encoded as the keyword "a", followed by a colon and a character string. If the Call Agent specifies a list of values, these values will be separated by a semicolon. For RTP, audio codecs SHALL be specified by using encoding names defined in the RTP AV Profile [4] or its replacement, or by encoding names registered with the IANA. Non-audio media registered as a MIME type MUST use the "<MIME type>/<MIME subtype>" form, as in "image/t38".

* The packetization period in milliseconds, encoded as the keyword "p", followed by a colon and a decimal number. If the Call Agent specifies a range of values, the range will be specified as two decimal numbers separated by a hyphen (as specified for the "ptime" parameter for SDP).

* The bandwidth in kilobits per second (1000 bits per second), encoded as the keyword "b", followed by a colon and a decimal number. If the Call Agent specifies a range of values, the range will be specified as two decimal numbers separated by a hyphen.

* The type of service parameter, encoded as the keyword "t", followed by a colon and the value encoded as two hexadecimal digits. When the connection is transmitted over an IP network, the parameters encode the 8-bit type of service value parameter of the IP header (a.k.a. DiffServ field). The left-most "bit" in the parameter corresponds to the least significant bit in the IP header.

* The echo cancellation parameter, encoded as the keyword "e", followed by a colon and the value "on" or "off".

* The gain control parameter, encoded as the keyword "gc", followed by a colon and a value which can be either the keyword "auto" or a decimal number (positive or negative) representing the number of decibels of gain.

* The silence suppression parameter, encoded as the keyword "s", followed by a colon and the value "on" or "off".

* The resource reservation parameter, encoded as the keyword "r", followed by a colon and the value "g" (guaranteed service), "cl" (controlled load) or "be" (best effort).

* The encryption key, encoded as the keyword "k" followed by a colon and a key specification, as defined for the parameter "K" in SDP (RFC 2327).

* The type of network, encoded as the keyword "nt" followed by a colon and the type of network encoded as the keyword "IN" (internet), "ATM", "LOCAL" (for a local connection), or possibly another type of network registered with the IANA as per SDP (RFC 2327).

* The resource reservation parameter, encoded as the keyword "r", followed by a colon and the value "g" (guaranteed service), "cl" (controlled load) or "be" (best effort).

The encoding of the first three attributes, when they are present, will be compatible with the SDP and RTP profiles. Note that each of the attributes is optional. When several attributes are present, they are separated by a comma.

Examples of local connection options are:

```
L: p:10, a:PCMU
L: p:10, a:G726-32
L: p:10-20, b:64
L: b:32-64, e:off
```

The set of Local Connection Options attributes can be extended in three different ways:

* Package Extension attributes (preferred)

* Vendor Extension attributes

* Other Extension attributes

Package Extension Local Connection Options attributes are defined in packages which provides the following benefits:

* A registration mechanism (IANA) for the package name.

* A separate name space for the attributes.

* A convenient grouping of the extensions.

* A simple way to determine support for them through auditing.

The package extension mechanism is the preferred extension method.

Vendor extension attributes are composed of an attribute name, and possibly followed by a colon and an attribute value. The attribute name MUST start with the two characters "x+", for a mandatory extension, or "x-", for a non-mandatory extension. If a gateway receives a mandatory extension attribute that it does not recognize, it MUST reject the command (error code 525 - unknown extension in LocalConnectionOptions, is RECOMMENDED).

Note that vendor extension attributes use an unmanaged name space, which implies a potential for name clashing. Vendors are consequently encouraged to include some vendor specific string, e.g., vendor name, in their vendor extensions.

Finally, for backwards compatibility with some existing implementations, MGCP allows for other extension attributes as well (see grammar in Appendix A). Note however, that these attribute extensions do not provide the package extension attribute benefits. Use of this mechanism for new extensions is discouraged.

3.2.2.11 MaxMGCPDatagram

The MaxMGCPDatagram can only be used for auditing, i.e., it is a valid RequestedInfo code and can be provided as a response parameter.

In responses, the MaxMGCPDatagram value is encoded as a string of up to nine decimal digits -- leading zeroes are not permitted. The following example illustrates the use of this parameter:

```
MD: 8100
```

3.2.2.12 ObservedEvents

The observed events parameter provides the list of events that have been observed. The event codes are the same as those used in the NotificationRequest. Events that have been accumulated according to the digit map may be grouped in a single string, however such practice is discouraged; they SHOULD be reported as lists of isolated events if other events were detected during the digit accumulation. Examples of observed events are:

```
O: L/hu
O: D/8295555T
O: D/8,D/2,D/9,D/5,D/5,L/hf,D/5,D/5,D/T
O: L/hf, L/hf, L/hu
```

3.2.2.13 PackageList

The Package List can only be used for auditing, i.e., it is a valid RequestedInfo code and can be provided as a response parameter.

The response parameter will consist of a comma separated list of packages supported. The first package returned in the list is the default package. Each package in the list consists of the package name followed by a colon, and the highest version number of the package supported.

An example of a package list is:

```
PL: L:1,G:1,D:0,FOO:2,T:1
```

Note that for backwards compatibility, support for this parameter is OPTIONAL.

3.2.2.14 QuarantineHandling

The quarantine handling parameter contains a list of comma separated keywords:

* The keyword "process" or "discard" to indicate the treatment of quarantined and observed events. If neither "process" or "discard" is present, "process" is assumed.

* The keyword "step" or "loop" to indicate whether at most one notification per NotificationRequest is allowed, or whether multiple notifications per NotificationRequest are allowed. If neither "step" nor "loop" is present, "step" is assumed.

The following values are valid examples:

```
Q: loop
Q: process
Q: loop,discard
```

3.2.2.15 ReasonCode

Reason codes are three-digit numeric values. The reason code is optionally followed by a white space and commentary, e.g.:

```
E: 900 Endpoint malfunctioning
```

A list of reason codes can be found in Section 2.5.

The set of reason codes can be extended through packages.

3.2.2.16 RequestedEvents

The RequestedEvents parameter provides the list of events that are requested. The event codes are described in Section 3.2.2.4.

Each event can be qualified by a requested action, or by a list of actions. The actions, when specified, are encoded as a list of keywords, enclosed in parenthesis and separated by commas. The codes for the various actions are:

Action	Code
Notify immediately	N
Accumulate	A
Treat according to digit map	D
Swap	S
Ignore	I
Keep Signal(s) active	K
Embedded Notification Request	E

When no action is specified, the default action is to notify the event. This means that, for example, ft and ft(N) are equivalent. Events that are not listed are ignored (unless they are persistent).

The digit-map action SHOULD only be specified for the digits, letters and interdigit timers in packages that define the encoding of digits, letters, and timers (including extension digit map letters).

The requested events list is encoded on a single line, with event/action groups separated by commas. Examples of RequestedEvents encodings are:

```
R: L/hu(N), L/hf(S,N)
R: L/hu(N), D/[0-9#T](D)
```

In the case of the "Embedded Notification Request" action, the embedded notification request parameters are encoded as a list of up to three parameter groups separated by commas. Each group starts by a one letter identifier, followed by a list of parameters enclosed between parentheses. The first optional parameter group, identified by the letter "R", is the value of the embedded RequestedEvents parameter. The second optional group, identified by the letter "S", is the embedded value of the SignalRequests parameter. The third

optional group, identified by the letter "D", is the embedded value of the DigitMap. (Note that some existing implementations and profiles may encode these three components in a different order. Implementers are encouraged to accept such encodings, but they SHOULD NOT generate them.)

If the RequestedEvents parameter is not present, the parameter will be set to a null value. If the SignalRequests parameter is not present, the parameter will be set to a null value. If the DigitMap is absent, the current value MUST be used. The following are valid examples of embedded requests:

```
R: L/hd(E(R(D/[0-9#T](D),L/hu(N)),S(L/dl),D([0-9].[#T])))
R: L/hd(E(R(D/[0-9#T](D),L/hu(N)),S(L/dl)))
```

Some events can be qualified by additional event parameters. Such event parameters will be separated by commas and enclosed within parentheses. Event parameters may be enclosed in double-quotes (in fact, some event parameters MUST be enclosed in double-quotes due to syntactic restrictions), in which case the quoted string itself is UTF-8 encoded. Please refer to Section 3.2.2.4 for additional detail on event parameters.

The following example shows the foobar event with an event parameter "epar":

```
R: X/foobar(N)(epar=2)
```

Notice that the Action was included even though it is the default Notify action - this is required by the grammar.

3.2.2.17 RequestedInfo

The RequestedInfo parameter contains a comma separated list of parameter codes, as defined in Section 3.2.2. For example, if one wants to audit the value of the NotifiedEntity, RequestIdentifier, RequestedEvents, SignalRequests, DigitMap, QuarantineHandling and DetectEvents parameters, the value of the RequestedInfo parameter will be:

```
F: N,X,R,S,D,Q,T
```

Note that extension parameters in general can be audited as well. The individual extension will define the auditing operation.

The capabilities request, in the AuditEndPoint command, is encoded by the parameter code "A", as in:

F: A

3.2.2.18 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest that triggered it. A RequestIdentifier is a hexadecimal string, at most 32 characters in length. RequestIdentifiers are compared as strings rather than numerical value. The string "0" is reserved for reporting of persistent events in the case where a NotificationRequest has not yet been received after restart.

3.2.2.19 ResponseAck

The response acknowledgement parameter is used to manage the "at-most-once" facility described in Section 3.5. It contains a comma separated list of "confirmed transaction-id ranges".

Each "confirmed transaction-id range" is composed of either one decimal number, when the range includes exactly one transaction, or two decimal numbers separated by a single hyphen, describing the lower and higher transaction identifiers included in the range.

An example of a response acknowledgement is:

K: 6234-6255, 6257, 19030-19044

3.2.2.20 RestartMethod

The RestartMethod parameter is encoded as one of the keywords "graceful", "forced", "restart", "disconnected" or "cancel-graceful" as for example:

RM: restart

The set of restart methods can be extended through packages.

3.2.2.21 SignalRequests

The SignalRequests parameter provides the name of the signal(s) that have been requested. Each signal is identified by a name, as described in Section 3.2.2.4.

Some signals, such as for example announcement or ADSI display, can be qualified by additional parameters, e.g.:

* the name and parameters of the announcement,

* the string that should be displayed.

Such parameters will be separated by commas and enclosed within parenthesis, as in:

```
S: L/adsi("123456 Francois Gerard")
S: A/ann(http://ann.example.net/no-such-number.au, 1234567)
```

When a quoted-string is provided, the string itself is UTF-8 encoded [20].

When several signals are requested, their codes are separated by a comma, as in:

```
S: L/adsi("123456 Your friend"), L/rg
```

Please refer to Section 3.2.2.4 for additional detail on signal parameters.

3.3 Format of response headers

The response header is composed of a response line, optionally followed by headers that encode the response parameters.

An example of a response header could be:

```
200 1203 OK
```

The response line starts with the response code, which is a three digit numeric value. The code is followed by a white space, and the transaction identifier. Response codes defined in packages (8xx) are followed by white space, a slash ("/") and the package name. All response codes may furthermore be followed by optional commentary preceded by a white space.

The following table describes the parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response. The letter M stands for mandatory, O for optional and F for forbidden. Unless otherwise specified, a parameter MUST NOT be present more than once. Note that the table only reflects the default for responses that have not defined any other behavior. If a response is received with a parameter that is either not understood or marked as forbidden, the offending parameter(s) MUST simply be ignored.

Parameter name	EP CF	CR CX	MD CX	DL CX	RQ NT	NT FY	AU EP	AU CX	RS IP
BearerInformation	F	F	F	F	F	F	O	F	F
CallId	F	F	F	F	F	F	F	O	F
Capabilities	F	F	F	F	F	F	O*	F	F
ConnectionId	F	O*	F	F	F	F	O*	F	F
ConnectionMode	F	F	F	F	F	F	F	O	F
Connection-Parameters	F	F	F	O*	F	F	F	O	F
DetectEvents	F	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	F	O	F	F
EventStates	F	F	F	F	F	F	O	F	F
LocalConnection-Options	F	F	F	F	F	F	F	O	F
MaxMGCPDatagram	F	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	F	O	O	O
ObservedEvents	F	F	F	F	F	F	O	F	F
QuarantineHandling	F	F	F	F	F	F	O	F	F
PackageList	O*	O*	O*	O*	O*	O*	O	O*	O*
ReasonCode	F	F	F	F	F	F	O	F	F
RequestIdentifier	F	F	F	F	F	F	O	F	F
ResponseAck	O*	O*	O*	O*	O*	O*	O*	O*	O*
RestartDelay	F	F	F	F	F	F	O	F	F
RestartMethod	F	F	F	F	F	F	O	F	F
RequestedEvents	F	F	F	F	F	F	O	F	F
RequestedInfo	F	F	F	F	F	F	F	F	F
SecondConnectionId	F	O	F	F	F	F	F	F	F
SecondEndpointId	F	O	F	F	F	F	F	F	F
SignalRequests	F	F	F	F	F	F	O	F	F
SpecificEndpointId	F	O	F	F	F	F	O*	F	F
LocalConnection-Descriptor	F	O*	O	F	F	F	F	O*	F
RemoteConnection-Descriptor	F	F	F	F	F	F	F	O*	F

Notes (*):

* The PackageList parameter is only allowed with return code 518 (unsupported package), except for AuditEndpoint, where it may also be returned if audited.

* The ResponseAck parameter MUST NOT be used with any other responses than a final response issued after a provisional response for the transaction in question. In that case, the presence of the ResponseAck parameter SHOULD trigger a Response Acknowledgement - any ResponseAck values provided will be ignored.

* In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter and a LocalConnectionDescriptor. It may also be followed a Specific-Endpoint-Id parameter, if the creation request was sent to a wildcarded Endpoint-Id. The connection-Id and LocalConnectionDescriptor parameter are marked as optional in the Table. In fact, they are mandatory with all positive responses, when a connection was created, and forbidden when the response is negative, and no connection was created.

* A LocalConnectionDescriptor MUST be transmitted with a positive response (code 200) to a CreateConnection. It MUST also be transmitted in response to a ModifyConnection command, if the modification resulted in a modification of the session parameters. The LocalConnectionDescriptor is encoded as a "session description", as defined in section 3.4. It is separated from the response header by an empty line.

* Connection-Parameters are only valid in a response to a non-wildcarded DeleteConnection command sent by the Call Agent.

* Multiple ConnectionId, SpecificEndpointId, and Capabilities parameters may be present in the response to an AuditEndpoint command.

* When several session descriptors are encoded in the same response, they are encoded one after each other, separated by an empty line. This is the case for example when the response to an audit connection request carries both a local session description and a remote session description, as in:

```
200 1203 OK
C: A3C47F21456789F0
N: [128.96.41.12]
L: p:10, a:PCMU;G726-32
M: sendrecv
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27,LA=48

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 1296 RTP/AVP 0

v=0
o=- 33343 346463 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0 96
a=rtpmap:96 G726-32/8000
```

In this example, according to the SDP syntax, each description starts with a "version" line, (v=...). The local description is always transmitted before the remote description. If a connection descriptor is requested, but it does not exist for the connection audited, that connection descriptor will appear with the SDP protocol version field only.

The response parameters are described for each of the commands in the following.

3.3.1 CreateConnection Response

In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter with a successful response (code 200). A LocalConnectionDescriptor is furthermore transmitted with a positive response. The LocalConnectionDescriptor is encoded as a "session description", as defined by SDP (RFC 2327). It is separated from the response header by an empty line, e.g.:

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96
a=rtpmap:96 G726-32/8000
```

When a provisional response has been issued previously, the final response SHOULD furthermore contain the Response Acknowledgement parameter (final responses issued by entities adhering to this specification will include the parameter, but older RFC 2705 implementations MAY not):

```
200 1204 OK
K:
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96
a=rtpmap:96 G726-32/8000
```

The final response SHOULD then be acknowledged by a Response Acknowledgement:

```
000 1204
```

3.3.2 ModifyConnection Response

In the case of a successful ModifyConnection message, the response line is followed by a LocalConnectionDescriptor, if the modification resulted in a modification of the session parameters (e.g., changing only the mode of a connection does not alter the session parameters). The LocalConnectionDescriptor is encoded as a "session description", as defined by SDP. It is separated from the response header by an empty line.

```
200 1207 OK

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

When a provisional response has been issued previously, the final response SHOULD furthermore contain the Response Acknowledgement parameter as in:

```
200 1207 OK
K:
```

The final response SHOULD then be acknowledged by a Response Acknowledgement:

```
000 1207 OK
```

3.3.3 DeleteConnection Response

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in Section 3.2.2.7.

```
250 1210 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

3.3.4 NotificationRequest Response

A successful NotificationRequest response does not include any additional response parameters.

3.3.5 Notify Response

A successful Notify response does not include any additional response parameters.

3.3.6 AuditEndpoint Response

In the case of a successful AuditEndPoint the response line may be followed by information for each of the parameters requested - each parameter will appear on a separate line. Parameters for which no

value currently exists, e.g., digit map, will still be provided but with an empty value. Each local endpoint name "expanded" by a wildcard character will appear on a separate line using the "SpecificEndPointId" parameter code, e.g.:

```
200 1200 OK
Z: aaln/1@rgw.whatever.net
Z: aaln/2@rgw.whatever.net
```

When connection identifiers are audited and multiple connections exist on the endpoint, a comma-separated list of connection identifiers SHOULD be returned as in:

```
200 1200 OK
I: FDE234C8, DFE233D1
```

Alternatively, multiple connection id parameter lines may be returned - the two forms should not be mixed although doing so does not constitute an error.

When capabilities are audited, the response may include multiple capabilities parameter lines as in:

```
200 1200 OK
A: a:PCMU;G728, p:10-100, e:on, s:off, t:1, v:L,
    m:sendonly;recvonly;sendrecv;inactive
A: a:G729, p:30-90, e:on, s:on, t:1, v:L,
    m:sendonly;recvonly;sendrecv;inactive;confrnce
```

Note: The carriage return for Capabilities shown above is present for formatting reasons only. It is not permissible in a real command encoding.

3.3.7 AuditConnection Response

In the case of a successful AuditConnection, the response may be followed by information for each of the parameters requested. Parameters for which no value currently exists will still be provided. Connection descriptors will always appear last and each will be preceded by an empty line, as for example:

```
200 1203 OK
C: A3C47F21456789F0
N: [128.96.41.12]
L: p:10, a:PCMU;G728
M: sendrecv
P: PS=622, OS=31172, PR=390, OR=22561, PL=5, JI=29, LA=50

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 96
a=rtpmap:96 G726-32/8000
```

If both a local and a remote connection descriptor are provided, the local connection descriptor will be the first of the two. If a connection descriptor is requested, but it does not exist for the connection audited, that connection descriptor will appear with the SDP protocol version field only ("v=0"), as for example:

```
200 1203 OK

v=0
```

3.3.8 RestartInProgress Response

A successful RestartInProgress response may include a NotifiedEntity parameter, but otherwise does not include any additional response parameters.

Also, a 521 response to a RestartInProgress MUST include a NotifiedEntity parameter with the name of another Call Agent to contact when the first Call Agent redirects the endpoint to another Call Agent as in:

```
521 1204 Redirect
N: CA-1@whatever.net
```

3.4 Encoding of the Session Description (SDP)

The session description (SDP) is encoded in conformance with the session description protocol, SDP. MGCP implementations are REQUIRED to be fully capable of parsing any conformant SDP message, and MUST send session descriptions that strictly conform to the SDP standard.

The general description and explanation of SDP parameters can be found in RFC 2327 (or its successor). In particular, it should be noted that the

* Origin ("o="),

* Session Name ("s="), and

* Time active ("t=")

are all mandatory in RFC 2327. While they are of little use to MGCP, they MUST be provided in conformance with RFC 2327 nevertheless. The following suggests values to be used for each of the fields, however the reader is encouraged to consult RFC 2327 (or its successor) for details:

Origin
o = <username> <session id> <version> <network type> <address type>
<address>

* The username SHOULD be set to hyphen ("-").

* The session id is RECOMMENDED to be an NTP timestamp as suggested in RFC 2327.

* The version is a version number that MUST increment with each change to the SDP. A counter initialized to zero or an NTP timestamp as suggested in RFC 2327 is RECOMMENDED.

* The network type defines the type of network. For RTP sessions the network type SHOULD be "IN".

* The address type defines the type of address. For RTP sessions the address type SHOULD be "IP4" (or "IP6").

* The address SHOULD be the same address as provided in the connection information ("c=") field.

Session Name
s = <session name>

The session name should be hyphen ("-").

Time active
t = <start time> <stop time>

* The start time may be set to zero.

* The stop time should be set to zero.

Each of the three fields can be ignored upon reception.

To further accommodate the extensibility principles of MGCP, implementations are ENCOURAGED to support the PINT "a=require" attribute - please refer to RFC 2848 for further details.

The usage of SDP actually depends on the type of session that is being established. Below we describe usage of SDP for an audio service using the RTP/AVP profile [4], or the LOCAL interconnect defined in this document. In case of any conflicts between what is described below and SDP (RFC 2327 or its successor), the SDP specification takes precedence.

3.4.1 Usage of SDP for an Audio Service

In a telephony gateway, we only have to describe sessions that use exactly one media, audio. The usage of SDP for this is straightforward and described in detail in RFC 2327.

The following is an example of an RFC 2327 conformant session description for an audio connection:

```
v=0
o=- A7453949499 0 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0 96
a=rtpmap:96 G726-32/8000
```

3.4.2 Usage of SDP for LOCAL Connections

When MGCP is used to set up internal connections within a single gateway, the SDP format is used to encode the parameters of that connection. The connection and media parameters will be used as follows:

* The connection parameter (c=) will specify that the connection is local, using the keyword "LOCAL" as network type, the keyword "EPN" (endpoint name) as address type, and the local name of the endpoint as the connection-address.

* The "m=audio" parameter will specify a port number, which will always be set to 0, the type of protocol, always set to the keyword LOCAL, and the type of encoding, using the same conventions used for the RTP AVP profile (RTP payload numbers). The type of encoding should normally be set to 0 (PCMU).

A session-level attribute identifying the connection MAY furthermore be present. This enables endpoints to support multiple LOCAL connections. Use of this attribute is OPTIONAL and indeed unnecessary for endpoints that only support a single LOCAL connection. The attribute is defined as follows:

a=MGCPlocalcx:<ConnectionID>
The MGCP Local Connection attribute is a session level only case-insensitive attribute that identifies the MGCP LOCAL connection, on the endpoint identified in the connection information, to which the SDP applies. The ConnectionId is a hexadecimal string containing at most 32 characters. The ConnectionId itself is case-insensitive. The MGCP Local Connection attribute is not subject to the charset attribute.

An example of a LOCAL session description could be:

```
v=0
o=- A7453949499 0 LOCAL EPN X35V3+A4/13
s=-
c=LOCAL EPN X35V3+A4/13
t=0 0
a=MGCPlocalcx:FDE234C8
m=audio 0 LOCAL 0
```

Note that the MGCP Local Connection attribute is specified at the session level and that it could have been omitted in case only a single LOCAL connection per endpoint is supported.

3.5 Transmission over UDP

MGCP messages are transmitted over UDP. Commands are sent to one of the IP addresses defined in the DNS for the specified endpoint. The responses are sent back to the source address (i.e., IP address and UDP port number) of the commands - the response may or may not arrive from the same address as the command was sent to.

When no port is specified for the endpoint, the commands MUST by default be sent:

* by the Call Agents, to the default MGCP port for gateways, 2427.

* by the Gateways, to the default MGCP port for Call Agents, 2727.

3.5.1 Providing the At-Most-Once Functionality

MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are retransmitted. Most MGCP commands are not idempotent. The state of the gateway would become unpredictable if, for example, CreateConnection commands were executed several times. The transmission procedures MUST thus provide an "at-most-once" functionality.

MGCP 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 being executed. The numerical value of transaction identifiers of incoming commands are compared to the transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction again, but simply resends the response. The remaining commands will be compared to the list of current transactions, i.e., transactions received previously which have not yet finished executing. If a match is found, the MGCP entity does not execute the transaction again, but a provisional response (Section 3.5.5) SHOULD be issued to acknowledge receipt of the command.

The procedure uses a long timer value, noted T-HIST in the following. The timer MUST be set larger than the maximum duration of a transaction, which MUST 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 T-HIST seconds after the response is issued, or when the gateway (or the Call Agent) receives a confirmation that the response has been received, through the "Response Acknowledgement". For transactions that are acknowledged through this attribute, the gateway SHALL keep a copy of the transaction-id (as opposed to the entire transaction response) for T-HIST 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.

3.5.2 Transaction Identifiers and Three Ways Handshake

Transaction identifiers are integer numbers in the range from 1 to 999,999,999 (both included). Call-agents may decide to use a specific number space for each of the gateways that they manage, or to use the same number space for all gateways that belong to some arbitrary group. Call agents may decide to share the load of managing a large gateway between several independent processes. These processes MUST then share the 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 MUST guarantee that unique transaction identifiers are allocated to all transactions that originate from a logical call agent, as defined in Section 4. Gateways can simply detect duplicate transactions by looking at the transaction identifier only.

The Response Acknowledgement Attribute can be found in any command. It carries a set of "confirmed transaction-id ranges" for final responses received - provisional responses MUST NOT be confirmed. A given response SHOULD NOT be confirmed in two separate messages.

MGCP entities MAY choose to delete the copies of the responses (but not the transaction-id) to transactions whose id is included in "confirmed transaction-id ranges" received in the Response Confirmation messages (command or response). They SHOULD then silently discard further commands from that entity when the transaction-id falls within these ranges, and the response was issued less than T-HIST seconds ago.

Entities MUST exercise due caution when acknowledging responses. In particular, a response SHOULD only be acknowledged if the response acknowledgement is sent to the same entity as the corresponding command (i.e., the command whose response is being acknowledged) was sent to.

Likewise, entities SHOULD NOT blindly accept a response acknowledgement for a given response. However it is considered safe to accept a response acknowledgement for a given response, when that response acknowledgement is sent by the same entity as the command that generated that response.

It should be noted, that use of response acknowledgments in commands (as opposed to the Response Acknowledgement response following a provisional response) is OPTIONAL. The benefit of using it is that it reduces overall memory consumption. However, in order to avoid large messages, implementations SHOULD NOT generate large response

acknowledgement lists. One strategy is to manage responses to commands on a per endpoint basis. A command for an endpoint can confirm a response to an older command for that same endpoint. Responses to commands with wildcarded endpoint names can be confirmed selectively with due consideration to message sizes, or alternatively simply not be acknowledged (unless the response explicitly required a Response Acknowledgement). Care must be taken to not confirm the same response twice or a response that is more than T-HIST seconds old.

The "confirmed transaction-id ranges" values SHALL NOT be used if more than T-HIST seconds have elapsed since the entity issued its last response to the other entity, or when an entity resumes operation. In this situation, commands MUST be accepted and processed, without any test on the transaction-id.

Commands that carry the "Response Acknowledgement attribute" may be transmitted in disorder. The union of the "confirmed transaction-id ranges" received in recent messages SHALL be retained.

3.5.3 Computing Retransmission Timers

It is the responsibility of the requesting entity to provide suitable time outs for all outstanding commands, and to retry commands when time outs have been exceeded. Furthermore, when repeated commands fail to be acknowledged, it is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending associations.

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 by measuring the time spent between the sending of a command and the return of the first response to the command. At a minimum, a retransmission strategy involving exponential backoff MUST be implemented. 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, RTO, in TCP, is set to the sum of the average delay plus N times the average deviation, where N is a constant. In MGCP, the maximum value of the timer SHOULD however be bounded, in order to guarantee that no repeated packet will be received by the gateways after T-HIST seconds. A suggested maximum value for RTO (RTO-MAX) is 4 seconds. Implementers SHOULD consider bounding the minimum value of this timer as well [19].

After any retransmission, the MGCP entity SHOULD do the following:

* It should double the estimated value of the acknowledgement delay for this transaction, T-DELAY.

* It should compute a random value, uniformly distributed between 0.5 T-DELAY and T-DELAY.

* It should set the retransmission timer (RTO) to the minimum of:
  - the sum of that random value and N times the average deviation,
  - RTO-MAX.

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.

Note that the estimators AAD and ADEV SHOULD NOT be updated for transactions that involve retransmissions. Also, the first new transmission following a successful retransmission SHOULD use the RTO for that last retransmission. If this transmission succeeds without any retransmissions, the AAD and ADEV estimators are updated and RTO is determined as usual again. See, e.g., [18] for further details.

3.5.4 Maximum Datagram Size, Fragmentation and Reassembly

MGCP messages being transmitted over UDP rely on IP for fragmentation and reassembly of large datagrams. The maximum theoretical size of an IP datagram is 65535 bytes. With a 20-byte IP header and an 8-byte UDP header, this leaves us with a maximum theoretical MGCP message size of 65507 bytes when using UDP.

However, IP does not require a host to receive IP datagrams larger than 576 bytes [21], which would provide an unacceptably small MGCP message size. Consequently, MGCP mandates that implementations MUST support MGCP datagrams up to at least 4000 bytes, which requires the

corresponding IP fragmentation and reassembly to be supported. Note, that the 4000 byte limit applies to the MGCP level. Lower layer overhead will require support for IP datagrams that are larger than this: UDP and IP overhead will be at least 28 bytes, and, e.g., use of IPSec will add additional overhead.

It should be noted, that the above applies to both Call Agents and endpoints. Call Agents can audit endpoints to determine if they support larger MGCP datagrams than specified above. Endpoints do currently not have a similar capability to determine if a Call Agent supports larger MGCP datagram sizes.

3.5.5 Piggybacking

There are cases when a Call Agent will want to send several messages at the same time to the same gateways, and vice versa. When several MGCP messages have to be sent in the same datagram, they MUST be separated by a line of text that contains a single dot, as in for example:

```
200 2005 OK
.
DLCX 1244 card23/21@tgw-7.example.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The piggybacked messages MUST be processed exactly as if they had been received one at a time in several separate datagrams. Each message in the datagram MUST be processed to completion and in order starting with the first message, and each command MUST be responded to. Errors encountered in a message that was piggybacked MUST NOT affect any of the other messages received in that datagram - each message is processed on its own.

Piggybacking can be used to achieve two things:

* Guaranteed in-order delivery and processing of messages.

* Fate sharing of message delivery.

When piggybacking is used to guarantee in-order delivery of messages, entities MUST ensure that this in-order delivery property is retained on retransmissions of the individual messages. An example of this is when multiple Notify's are sent using piggybacking (as described in Section 4.4.1).

Fate sharing of message delivery ensures that either all the messages are delivered, or none of them are delivered. When piggybacking is used to guarantee this fate-sharing, entities MUST also ensure that this property is retained upon retransmission. For example, upon receiving a Notify from an endpoint operating in lockstep mode, the Call Agent may wish to send the response and a new NotificationRequest command in a single datagram to ensure message delivery fate-sharing of the two.

3.5.6 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.

Gateways (and Call Agents) that can predict that a transaction will require a long execution time SHOULD send a provisional response with response code 100. As a guideline, a transaction that requires external communication to complete, e.g., network resource reservation, SHOULD issue a provisional response. Furthermore entities SHOULD send a provisional response if they receive a repetition of a transaction that has not yet finished executing.

Gateways (or Call Agents) that start building up queues of transactions to be executed may send a provisional response with response code 101 to indicate this (see Section 4.4.8 for further details).

Pure transactional semantics would imply, that provisional responses SHOULD NOT return any other information than the fact that the transaction is currently executing, however an optimistic approach allowing some information to be returned enables a reduction in the delay that would otherwise be incurred in the system.

In order to reduce the delay in the system, it is RECOMMENDED to include a connection identifier and session description in a 100 provisional response to the CreateConnection command. If a session description would be returned by the ModifyConnection command, the session description SHOULD be included in the provisional response here as well. If the transaction completes successfully, the information returned in the provisional response MUST be repeated in the final response. It is considered a protocol error not to repeat this information or to change any of the previously supplied information in a successful response. If the transaction fails, an error code is returned - the information returned previously is no longer valid.

A currently executing CreateConnection or ModifyConnection transaction MUST be cancelled if a DeleteConnection command for the endpoint is received. In that case, a final response for the cancelled transaction SHOULD still be returned automatically (error code 407 - transaction aborted, is RECOMMENDED), and a final response for the cancelled transaction MUST be returned if a retransmission of the cancelled transaction is detected (see also Section 4.4.4).

MGCP entities that receive a provisional response SHALL switch to a longer repetition timer (LONGTRAN-TIMER) for that transaction. The purpose of this timer is primarily to detect processing failures. The default value of LONGTRAN-TIMER is 5 seconds, however the provisioning process may alter this. Note, that retransmissions MUST still satisfy the timing requirements specified in Section 3.5.1 and 3.5.3. Consequently LONGTRAN-TIMER MUST be smaller than T-HIST (it should in fact be considerably smaller). Also, entities MUST NOT let a transaction run forever. A transaction that is timed out by the entity SHOULD return error code 406 (transaction time-out). Per the definition of T-HIST (Section 3.5.1), the maximum transaction execution time is smaller than T-HIST (in a network with low delay, it can reasonably safely be approximated as T-HIST minus T-MAX), and a final response should be received no more than T-HIST seconds after the command was sent initially. Nevertheless, entities SHOULD wait for 2*T-HIST seconds before giving up on receiving a final response. Retransmission of the command MUST still cease after T-MAX seconds though. If a response is not received, the outcome of the transaction is not known. If the entity sending the command was a gateway, it now becomes "disconnected" and SHALL initiate the "disconnected" procedure (see Section 4.4.7).

When the transaction finishes execution, the final response is sent and the by now obsolete provisional response is deleted. In order to ensure rapid detection of a lost final response, final responses issued after provisional responses for a transaction SHOULD be acknowledged (unfortunately older RFC 2705 implementations may not do this, which is the only reason it is not an absolute requirement).

The endpoint SHOULD therefore include an empty "ResponseAck" parameter in those, and only those, final responses. The presence of the "ResponseAck" parameter in the final response SHOULD trigger a "Response Acknowledgement" response to be sent back to the endpoint. The Response Acknowledgement" response will then include the transaction-id of the response it acknowledges in the response header. Note that, for backwards compatibility, entities cannot depend on receiving such a "response acknowledgement", however it is strongly RECOMMENDED to support this behavior, as excessive delays in case of packet loss as well as excessive retransmissions may occur otherwise.

Receipt of a "Response Acknowledgement" response is subject to the same time-out and retransmission strategies and procedures as responses to commands, i.e., the sender of the final response will retransmit it if a "Response Acknowledgement" is not received in time. For backwards compatibility, failure to receive a "response acknowledgement" SHOULD NOT affect the roundtrip time estimates for subsequent commands, and furthermore MUST NOT lead to the endpoint becoming "disconnected". The "Response Acknowledgment" response is never acknowledged.

4. States, Failover and Race Conditions

In order to implement proper call signaling, the Call Agent must keep track of the state of the endpoint, and the gateway must make sure that events are properly notified to the Call Agent. Special conditions exist when the gateway or the Call Agent are restarted: the gateway must be redirected to a new Call Agent during "failover" procedures, the Call Agent must take special action when the gateway is taken offline, or restarted.

4.1 Failover Assumptions and Highlights

The following protocol highlights are important to understanding Call Agent fail-over mechanisms:

* Call Agents are identified by their domain name (and optional port), not their network addresses, and several addresses can be associated with a domain name.

* An endpoint has one and only one Call Agent associated with it at any given point in time. The Call Agent associated with an endpoint is the current value of the "notified entity". The "notified entity" determines where the gateway will send it's commands. If the "notified entity" does not include a port number, the default Call Agent port number (2727) is assumed.

* NotifiedEntity is a parameter sent by the Call Agent to the gateway to set the "notified entity" for the endpoint.

* The "notified entity" for an endpoint is the last value of the NotifiedEntity parameter received for this endpoint. If no explicit NotifiedEntity parameter has ever been received, the "notified entity" defaults to a provisioned value. If no value was provisioned or an empty NotifiedEntity parameter was provided (both strongly discouraged) thereby making the "notified entity" empty, the "notified entity" is set to the source address of the last non-audit command for the endpoint. Thus auditing will not change the "notified entity".

* Responses to commands are sent to the source address of the command, regardless of the current "notified entity". When a Notify message needs to be piggybacked with the response, the datagram is still sent to the source address of the new command received, regardless of the current "notified entity".

The ability for the "notified entity" to resolve to multiple network addresses, allows a "notified entity" to represent a Call Agent with multiple physical interfaces on it and/or a logical Call Agent made up of multiple physical systems. The order of network addresses when a DNS name resolves to multiple addresses is non-deterministic so Call Agent fail-over schemes MUST NOT depend on any order (e.g., a gateway MUST be able to send a "Notify" to any of the resolved network addresses). On the other hand, the system is likely to be most efficient if the gateway sends commands to the interface with which it already has a current association. It is RECOMMENDED that gateways use the following algorithm to achieve that goal:

* If the "notified entity" resolves to multiple network addresses, and the source address of the request is one of those addresses, that network address is the preferred destination address for commands.

* If on the other hand, the source address of the request is not one of the resolved addresses, the gateway must choose one of the resolved addresses for commands.

* If the gateway fails to contact the network address chosen, it MUST try the alternatives in the resolved list as described in Section 4.3.

If an entire Call Agent becomes unavailable, the endpoints managed by that Call Agent will eventually become "disconnected". The only way for these endpoints to become connected again is either for the failed Call Agent to become available, or for a backup call agent to contact the affected endpoints with a new "notified entity".

When a backup Call Agent has taken over control of a group of endpoints, it is assumed that the failed Call Agent will communicate and synchronize with the backup Call Agent in order to transfer control of the affected endpoints back to the original Call Agent. Alternatively, the failed Call Agent could simply become the backup Call Agent.

We should note that handover conflict resolution between separate CA's is not in place - we are relying strictly on the CA's knowing what they are doing and communicating with each other (although AuditEndpoint can be used to learn about the current "notified entity"). If this is not the case, unexpected behavior may occur.

Note that as mentioned earlier, the default "notified entity" is provisioned and may include both domain name and port. For small gateways, provisioning may be done on a per endpoint basis. For much larger gateways, a single provisioning element may be provided for multiple endpoints or even for the entire gateway itself. In either case, once the gateway powers up, each endpoint MUST have its own "notified entity", so provisioned values for an aggregation of endpoints MUST be copied to the "notified entity" for each endpoint in the aggregation before operation proceeds. Where possible, the RestartInProgress command on restart SHOULD be sent to the provisioned "notified entity" based on an aggregation that allows the "all of" wild-card to be used. This will reduce the number of RestartInProgress messages.

Another way of viewing the use of "notified entity" is in terms of associations between gateways and Call Agents. The "notified entity" is a means to set up that association, and governs where the gateway will send commands to. Commands received by the gateway however may come from any source. The association is initially provisioned with a provisioned "notified entity", so that on power up RestartInProgress and persistent events that occur prior to the first NotificationRequest from Call Agents will be sent to the provisioned Call Agent. Once a Call Agent makes a request, however it may include the NotifiedEntity parameter and set up a new association. Since the "notified entity" persists across calls, the association remains intact until a new "notified entity" is provided.

4.2 Communicating with Gateways

Endpoint names in gateways include a local name indicating the specific endpoint and a domain name indicating the host/gateway where the endpoint resides. Gateways may have several interfaces for redundancy.

In gateways that have routing capability, the domain name may resolve to a single network address with internal routing to that address from any of the gateway's interfaces. In others, the domain name may resolve to multiple network addresses, one for each interface. In the latter case, if a Call Agent fails to contact the gateway on one of the addresses, it MUST try the alternates.

4.3 Retransmission, and Detection of Lost Associations:

The media gateway control protocol is organized as a set of transactions, each of which is composed of a command and a response, commonly referred to as an acknowledgement. The MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are retransmitted. MGCP entities MUST 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 T-HIST seconds, and a list of the transactions that have not yet finished executing.

The transaction identifiers of incoming commands are compared to the transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction, but simply repeats the response. If a match to a previously responded to transaction is not found, the transaction identifier of the incoming command is compared to the list of transactions that have not yet finished executing. If a match is found, the MGCP entity does not execute the transaction again, but SHOULD simply send a provisional response - a final response will be provided when the execution of the command is complete (see Section 3.5.6 for further detail).

The repetition mechanism is used to guard against four 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 Call Agent becomes unavailable,

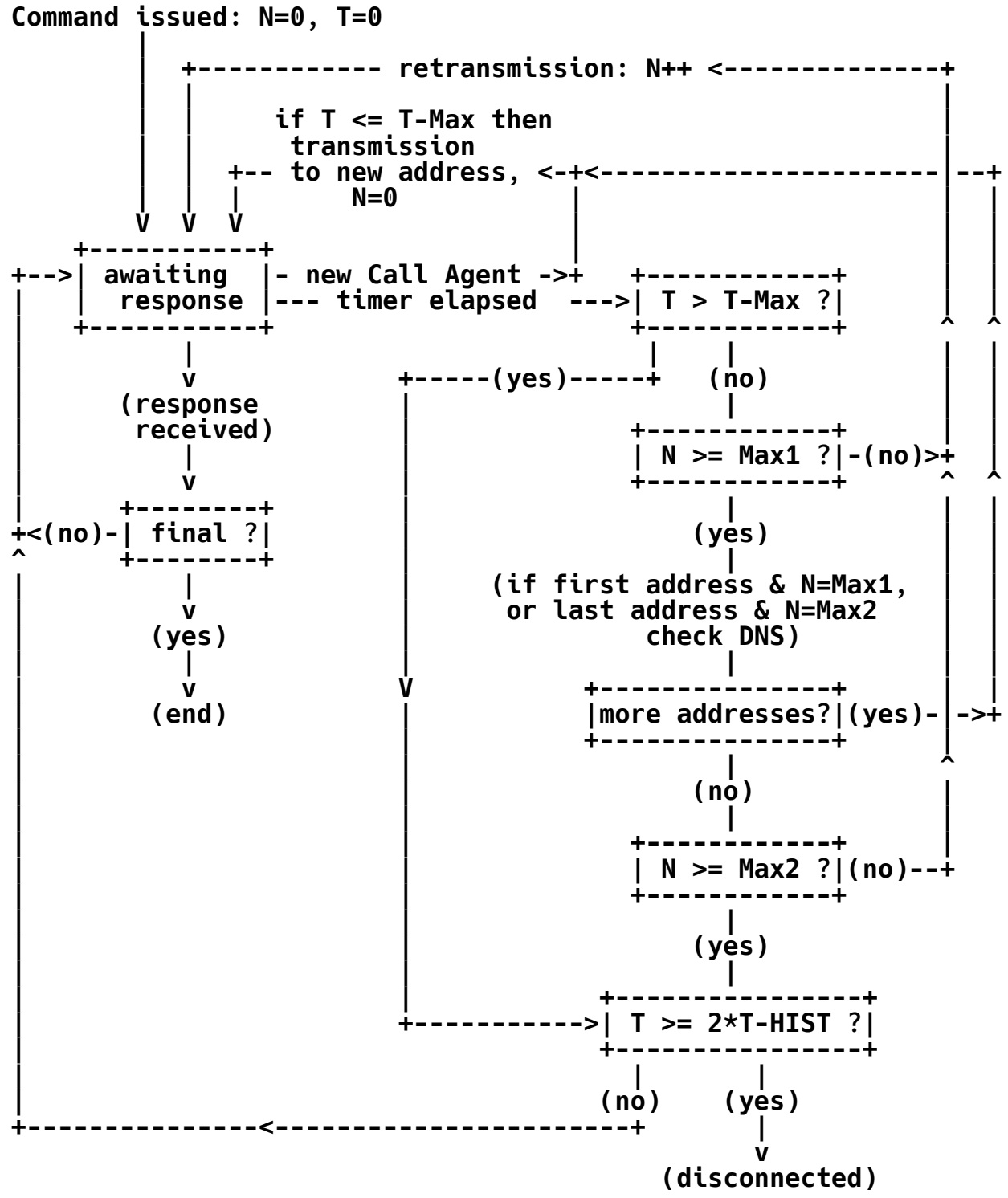
* Call Agent failure, when for example an entire Call Agent becomes unavailable,

* failover, when a new Call Agent is "taking over" transparently.

The elements 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 very low, typically less than 1%. If a Call Agent or a gateway has to repeat a message more than a few times, it is very legitimate to assume that something other than a transmission error is occurring. For example, given a loss rate of 1%, the probability that 5 consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a Call Agent that processes 1,000 transactions per second. (Indeed, the number of retransmissions 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 we will call "Max2". Max2 MUST be set to a larger value than Max1.

The MGCP retransmission algorithm is illustrated in the Figure below and explained further in the following:


Command issued: N=0, T=0
retransmission: N++
if T <= T-Max then transmission to new address, N=0
awaiting response
new Call Agent
timer elapsed
T > T-Max ?
(yes)
(no)
N >= Max1 ?
(no)
(yes)
(if first address & N=Max1, or last address & N=Max2 check DNS)
more addresses?
(yes)
(no)
N >= Max2 ?
(no)
(yes)
T >= 2*T-HIST ?
(no)
(yes)
(disconnected)
(response received)
final ?
(no)
(yes)
(end)

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after re-transmitting 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 as follows:

* We require that the gateway always checks for the presence of a new Call Agent. It can be noticed either by:

  - receiving a command where the NotifiedEntity points to the new Call Agent, or

  - receiving a redirection response pointing to a new Call Agent.

  If a new Call Agent is detected, the gateway MUST start retransmitting outstanding commands for the endpoint(s) redirected to that new Call Agent. Responses to new or old commands are still transmitted to the source address of the command.

* 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, then retransmissions MUST cease. If more than 2*T-HIST has elapsed, then the endpoint becomes disconnected.

* If the number of repetitions for this Call Agent is equal to "Max1", and its domain name was not resolved recently (e.g., within the last 5 seconds or otherwise provisioned), and it is not in the process of being resolved, then the gateway MAY actively query the domain name server in order to detect the possible change of the Call Agent interfaces. Note that the first repetition is the second transmission.

* The gateway may have learned several IP addresses for the call agent. If the number of repetitions for this IP address is greater than or equal to "Max1" and lower than "Max2", and there are more addresses that have not been tried, then the gateway MUST direct the retransmissions to alternate addresses. Also, receipt of explicit network notifications such as, e.g., ICMP network, host, protocol, or port unreachable SHOULD lead the gateway to try alternate addresses (with due consideration to possible security issues).

* If there are no more interfaces to try, and the number of repetitions for this address is Max2, then the gateway SHOULD contact the DNS one more time to see if any other interfaces have become available, unless the domain name was resolved recently (e.g., within the last 5 seconds or otherwise provisioned), or it is already in the process of being resolved. If there still are no more interfaces to try, the gateway is then disconnected and MUST initiate the "disconnected" procedure (see Section 4.4.7).

In order to automatically adapt to network load, MGCP 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 (T) elapsed since the sending of the initial datagram is no greater than T-MAX. If more than T-MAX time has elapsed, retransmissions MUST cease. If more than 2*T-HIST time has elapsed, the endpoint becomes disconnected. The value T-MAX is related to the T-HIST value: the T-HIST value MUST be greater than or equal to T-MAX plus the maximum propagation delay in the network.

The default value for T-MAX is 20 seconds. Thus, if the assumed maximum propagation delay is 10 seconds, then responses to old transactions would have to be kept for a period of at least 30 seconds. The importance of having the sender and receiver agree on these values cannot be overstated.

The default value for Max1 is 5 retransmissions and the default value for Max2 is 7 retransmissions. Both of these values may be altered by the provisioning process.

The provisioning process MUST be able to disable one or both of the Max1 and Max2 DNS queries.

4.4 Race Conditions

MGCP deals with race conditions through the notion of a "quarantine list" and through explicit detection of desynchronization, e.g., for mismatched hook state due to glare for an endpoint.

MGCP does not assume that the transport mechanism will maintain the order of commands and responses. This may cause race conditions, that may be obviated through a proper behavior of the Call Agent. (Note that some race conditions are inherent to distributed systems; they would still occur, even if the commands were transmitted in strict order.)

In some cases, many gateways may decide to restart operation at the same time. This may occur, for example, if an area loses power or transmission capability during an earthquake or an ice storm. When power and transmission are reestablished, many gateways may decide to send "RestartInProgress" commands simultaneously, leading to very unstable operation.

4.4.1 Quarantine List

MGCP controlled gateways will receive "notification requests" that ask them to watch for a list of "events". The protocol elements that determine the handling of these events are the "Requested Events" list, the "Digit Map", the "Quarantine Handling", and the "Detect Events" list.

When the endpoint is initialized, the requested events list only consists of persistent events for the endpoint, and the digit map is assumed empty. At this point, the endpoint MAY use an implicit NotificationRequest with the reserved RequestIdentifier zero ("0") to detect and report a persistent event, e.g., off-hook. A pre-existing off-hook condition MUST here result in the off-hook event being generated as well.

The endpoint awaits the reception of a NotificationRequest command, after which the gateway starts observing the endpoint for occurrences of the events mentioned in the list, including persistent events.

The events are examined as they occur. The action that follows is determined by the "action" parameter associated with the event in the list of requested events, and also by the digit map. The events that are defined as "accumulate" or "accumulate according to digit map" are accumulated in a list of events, the events that are marked as "accumulate according to the digit map" will additionally be accumulated in the "current dial string". This will go on until one event is encountered that triggers a notification which will be sent to the current "notified entity".

The gateway, at this point, will transmit the Notify command and will place the endpoint in a "notification" state. As long as the endpoint is in this notification state, the events that are to be detected on the endpoint are stored in a "quarantine" buffer (FIFO)

for later processing. The events are, in a sense, "quarantined". All events that are specified by the union of the RequestedEvents parameter and the most recently received DetectEvents parameter or, in the absence of the latter, all events that are referred to in the RequestedEvents, SHALL be detected and quarantined, regardless of the action associated with the event. Persistent events are here viewed as implicitly included in RequestedEvents. If the quarantine buffer reaches the capacity of the endpoint, a Quarantine Buffer Overflow event (see Appendix B) SHOULD be generated (when this event is supported, the endpoint MUST ensure it has capacity to include the event in the quarantine buffer). Excess events will now be discarded.

The endpoint exits the "notification state" when the response (whether success or failure) to the Notify command is received. The Notify command may be retransmitted in the "notification state", as specified in Section 3.5 and 4. If the endpoint is or becomes disconnected (see Section 4.3) during this, a response to the Notify command will never be received. The Notify command is then lost and hence no longer considered pending, yet the endpoint is still in the "notification state". Should that occur, completion of the disconnected procedure specified in Section 4.4.7 SHALL then lead the endpoint to exit the "notification state".

When the endpoint exits the "notification state" it resets the list of observed events and the "current dial string" of the endpoint to a null value.

Following that point, the behavior of the gateway depends on the value of the QuarantineHandling parameter in the triggering NotificationRequest command:

If the Call Agent had specified, that it expected at most one notification in response to the notification request command, then the gateway SHALL simply keep on accumulating events in the quarantine buffer until it receives the next notification request command.

If, however, the gateway is authorized to send multiple successive Notify commands, it will proceed as follows. When the gateway exits the "notification state", it resets the list of observed events and the "current dial string" of the endpoint to a null value and starts processing the list of quarantined events, using the already received list of requested events and digit map. When processing these events, the gateway may encounter an event which triggers a Notify command to be sent. If that is the case, the gateway can adopt one of the two following behaviors:

* it can immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included, leaving the unprocessed events in the quarantine buffer,

* or it can attempt to empty the quarantine buffer and transmit a single Notify command reporting several sets of events (in a single list of observed events) and possibly several dial strings. The "current dial string" is reset to a null value after each triggering event. The events that follow the last triggering event are left in the quarantine buffer.

If the gateway transmits a Notify command, the endpoint will reenter and remain in the "notification state" until the acknowledgement is received (as described above). If the gateway does not find a quarantined event that triggers a Notify command, it places the endpoint in a normal state. Events are then processed as they come, in exactly the same way as if a Notification Request command had just been received.

A gateway may receive at any time a new Notification Request command for the endpoint, including the case where the endpoint is disconnected. Activating an embedded Notification Request is here viewed as receiving a new Notification Request as well, except that the current list of ObservedEvents remains unmodified rather than being processed again. When a new notification request is received in the notification state, the gateway SHALL ensure that the pending Notify is received by the Call Agent prior to a new Notify (note that a Notify that was lost due to being disconnected, is no longer considered pending). It does so by using the "piggybacking" functionality of the protocol. The messages will then be sent in a single packet to the current "notified entity". The steps involved are the following:

a) the gateway sends a response to the new notification request.

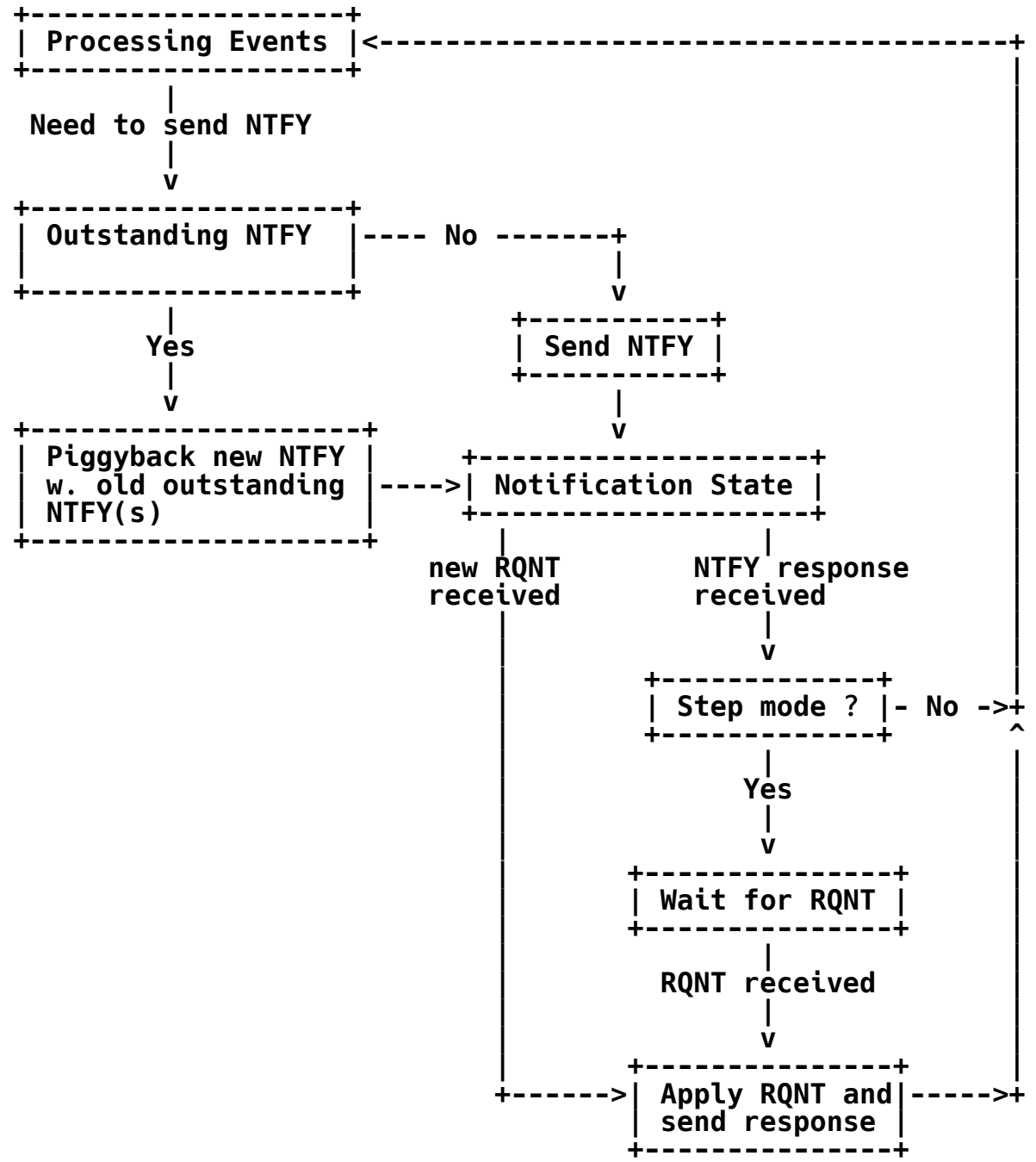
b) the endpoint is then taken out of the "notification state" without waiting for the acknowledgement of the pending Notify command.

c) a copy of the unacknowledged Notify command is kept until an acknowledgement is received. If a timer elapses, the Notify will be retransmitted.

d) If the gateway has to transmit a new Notify before the previous Notify(s) is acknowledged, it constructs a packet that piggybacks a repetition of the old Notify(s) and the new Notify (ordered by age with the oldest first). This datagram will be sent to the current "notified entity".

f) Gateways that cannot piggyback several messages in the same datagram and hence guarantee in-order delivery of two (or more) Notify's SHALL leave the endpoint in the "notification" state as long as the last Notify is not acknowledged.

The procedure is illustrated by the following diagram:



Gateways may also attempt to deliver the pending Notify prior to a successful response to the new NotificationRequest by using the "piggybacking" functionality of the protocol. This was in fact required behavior in RFC 2705, however there are several complications in doing this, and the benefits are questionable. In particular, the RFC 2705 mechanism did not guarantee in-order delivery of Notify's and responses to NotificationRequests in general, and hence Call Agents had to handle out-of-order delivery of these messages anyway. The change to optional status is thus backwards compatible while greatly reducing complexity.

After receiving the Notification Request command, the requested events list and digit map (if a new one was provided) are replaced by the newly received parameters, and the current dial string is reset to a null value. Furthermore, when the Notification Request was received in the "notification state", the list of observed events is reset to a null value. The subsequent behavior is conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events (and observed events which in this case is now an empty list), should be discarded, in which case they will be. If the parameter specifies that the quarantined (and observed) events are to be processed, the gateway will start processing the list of quarantined (and observed) events, using the newly received list of requested events and digit map (if provided). When processing these events, the gateway may encounter an event which requires a Notify command to be sent. If that is the case, the gateway will immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included leaving the unprocessed events in the quarantine buffer, and will enter the "notification state".

A new notification request may be received while the gateway has accumulated events according to the previous notification request, but has not yet detected a notification-triggering events, i.e., the endpoint is not in the "notification state". The handling of not-yet-notified events is determined, as with the quarantined events, by the quarantine handling parameter:

* If the quarantine-handling parameter specifies that quarantined events shall be ignored, the observed events list is simply reset.

* If the quarantine-handling parameter specifies that quarantined events shall be processed, the observed event list is transferred to the quarantined event list. The observed event list is then reset, and the quarantined event list is processed.

Call Agents controlling endpoints in lockstep mode SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggybacking mechanism.

4.4.2 Explicit Detection

A key element of the state of several endpoints is the position of the hook. A race condition may occur when the user decides to go off-hook before the Call Agent has the time to ask the gateway to notify an off-hook event (the "glare" condition well known in telephony), or if the user goes on-hook before the Call Agent has the time to request the event's notification.

To avoid this race condition, the gateway MUST check the condition of the endpoint before acknowledging a NotificationRequest. It MUST return an error:

1. If the gateway is requested to notify an "off-hook" transition while the phone is already off-hook, (error code 401 - phone off hook)

2. If the gateway is requested to notify an "on-hook" or "flash hook" condition while the phone is already on-hook (error code 402 - phone on hook).

Additionally, individual signal definitions can specify that a signal will only operate under certain conditions, e.g., ringing may only be possible if the phone is already off-hook. If such prerequisites exist for a given signal, the gateway MUST return the error specified in the signal definition if the prerequisite is not met.

It should be noted, that the condition check is performed at the time the notification request is received, whereas the actual event that caused the current condition may have either been reported, or ignored earlier, or it may currently be quarantined.

The other state variables of the gateway, such as the list of RequestedEvents or list of requested signals, are entirely replaced after each successful NotificationRequest, which prevents any long term discrepancy between the Call Agent and the gateway.

When a NotificationRequest is unsuccessful, whether it is included in a connection-handling command or not, the gateway MUST simply continue as if the command had never been received. As all other transactions, the NotificationRequest MUST operate as an atomic transaction, thus any changes initiated as a result of the command MUST be reverted.

Another race condition may occur when a Notify is issued shortly before the reception by the gateway of a NotificationRequest. The RequestIdentifier is used to correlate Notify commands with NotificationRequest commands thereby enabling the Call Agent to determine if the Notify command was generated before or after the gateway received the new NotificationRequest. This is especially important to avoid deadlocks in "step" mode.

4.4.3 Transactional Semantics

As the potential transaction completion times increase, e.g., due to external resource reservations, a careful definition of the transactional semantics becomes increasingly important. In particular the issue of race conditions, e.g., as it relates to hook-state, must be defined carefully.

An important point to consider is, that the status of a pre-condition (e.g., hook-state) may in fact change between the time a transaction starts and the time it either completes successfully (transaction commit) or fails. In general, we can say that the successful execution of a transaction depends on one or more pre-conditions where the status of one or more of the pre-conditions may change dynamically between the transaction start and transaction commit.

The simplest semantics for this is simply to require that all pre-conditions be met from the time the transaction is initiated until the transaction commits. If any pre-condition is not met before the completion of the transaction, the transaction will also fail.

As an example, consider a transaction that includes a request for the "off-hook" event. When the transaction is initiated the phone is "on-hook" and this pre-condition is therefore met. If the hook-state changes to "off-hook" before the transaction completes, the pre-condition is no longer met, and the transaction therefore immediately fails.

Finally, we need to consider the point in time when a new transaction takes effect and endpoint processing according to an old transaction stops. For example, assume that transaction T1 has been executed successfully and event processing is currently being done according to transaction T1. Now we receive a new transaction T2 specifying new event processing (for example a CreateConnection with an encapsulated NotificationRequest). Since we don't know whether T2 will complete successfully or not, we cannot start processing events according to T2 until the outcome of T2 is known. While we could suspend all event processing until the outcome of T2 is known, this would make for a less responsive system and hence SHOULD NOT be done. Instead, when a new transaction Ty is received and Ty modifies

processing according to an old transaction Tx, processing according to Tx SHOULD remain active for as long as possible, until a successful outcome of Ty is known to occur. If Ty fails, then processing according to Tx will of course continue as usual. Any changes incurred by Ty logically takes effect when Ty commits. Thus, if the endpoint was in the notification state when Ty commits, and Ty contained a NotificationRequest, the endpoint will be taken out of the notification state when Ty commits. Note that this is independent of whether the endpoint was in the notification state when Ty was initiated. For example, a Notify could be generated due to processing according to Tx between the start and commit of Ty. If the commit of Ty leads to the endpoint entering the notification state, a new NotificationRequest (Tz) is needed to exit the notification state. This follows from the fact that transaction execution respects causal order.

Another related issue is the use of wildcards, especially the "all of" wildcard, which may match more than one endpoint. When a command is requested, and the endpoint identifier matches more than one endpoint, transactional semantics still apply. Thus, the command MUST either succeed for all the endpoints, or it MUST fail for all of them. A single response is consequently always issued.

4.4.4 Ordering of Commands, and Treatment of Misorder

MGCP does not mandate that the underlying transport protocol guarantees in-order delivery of commands to a gateway or an endpoint. 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 Call Agent after the transmission of a new Notification Request command,

* If a new NotificationRequest is transmitted before a previous one is acknowledged, there is no guarantee that the previous one will not be received and executed after the new one.

Call Agents that want to guarantee consistent operation of the endpoints can use the following rules:

1) When a gateway handles several endpoints, commands pertaining to the different endpoints can be sent in parallel, for example following a model where each endpoint is controlled by its own process or its own thread.

2) When several connections are created on the same endpoint, commands pertaining to different connections can be sent in parallel.

3) On a given connection, there should normally be only one outstanding command (create or modify). However, a DeleteConnection command can be issued at any time. In consequence, a gateway may sometimes receive a ModifyConnection command that applies to a previously deleted connection. Such commands will fail, and an error code MUST be returned (error code 515 - incorrect connection-id, is RECOMMENDED).

4) On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter MUST be used to correlate Notify commands with the triggering notification request.

5) In some cases, an implicitly or explicitly wildcarded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The Call Agent should individually delete all connections whose completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wild-carding SHOULD NOT be sent until the wild-carded DeleteConnection command is acknowledged.

6) When commands are embedded within each other, sequencing requirements for all commands must be adhered to. For example a Create Connection command with a Notification Request in it must adhere to the sequencing requirements associated with both CreateConnection and NotificationRequest at the same time.

7) AuditEndpoint and AuditConnection are not subject to any sequencing requirements.

8) RestartInProgress MUST always be the first command sent by an endpoint as defined by the restart procedure. Any other command or non-restart response (see Section 4.4.6), except for responses to auditing, MUST be delivered after this RestartInProgress command (piggybacking allowed).

9) When multiple messages are piggybacked in a single packet, the messages are always processed in order.

10) On a given endpoint, there should normally be only one outstanding EndpointConfiguration command at any time.

Gateways MUST NOT make any assumptions as to whether Call Agents follow these rules or not. Consequently gateways MUST always respond to commands, regardless of whether they adhere to the above rules or not. To ensure consistent operation, gateways SHOULD behave as specified below when one or more of the above rules are not followed:

* Where a single outstanding command is expected (ModifyConnection, NotificationRequest, and EndpointConfiguration), but the same command is received in a new transaction before the old finishes executing, the gateway SHOULD fail the previous command. This includes the case where one or more of the commands were encapsulated. The use of error code 407 (transaction aborted) is RECOMMENDED.

* If a ModifyConnection command is received for a pending CreateConnection command, the ModifyConnection command SHOULD simply be rejected. The use of error code 400 (transient error) is RECOMMENDED. Note that this situation constitutes a Call Agent programming error.

* If a DeleteConnection command is received for a pending CreateConnection or ModifyConnection command, the pending command MUST be aborted. The use of error code 407 (transaction aborted) is RECOMMENDED.

Note, that where reception of a new command leads to aborting an old command, the old command SHOULD be aborted regardless of whether the new command succeeds or not. For example, if a ModifyConnection command is aborted by a DeleteConnection command which itself fails due to an encapsulated NotificationRequest, the ModifyConnection command is still aborted.

4.4.5 Endpoint Service States

As described earlier, endpoints configured for operation may be either in-service or out-of-service. The actual service-state of the endpoint is reflected by the combination of the RestartMethod and RestartDelay parameters, which are sent with RestartInProgress commands (Section 2.3.12) and furthermore may be audited in AuditEndpoint commands (Section 2.3.10).

The service-state of an endpoint affects how it processes a command. An endpoint in-service MUST process any command received, whereas an endpoint that is out-of-service MUST reject non-auditing commands, but SHOULD process auditing commands if possible. For backwards compatibility, auditing commands for an out-of-service endpoint may alternatively be rejected as well. Any command rejected due to an endpoint being out-of-service SHOULD generate error code 501 (endpoint not ready/out-of-service).

Note that (per Section 2.1.2), unless otherwise specified for a command, endpoint names containing the "any of" wildcard only refer to endpoints in-service, whereas endpoint names containing the "all of" wildcard refer to all endpoints, regardless of service state.

The above relationships are illustrated in the table below which shows the current service-states and gateway processing of commands as a function of the RestartInProgress command sent and the response (if any) received to it. The last column also lists (in parentheses) the RestartMethod to be returned if audited:

Restart-Method	Restart-Delay	2xx received ?	Service-State	Response to new command
graceful	zero	Yes/No	In	non-audit: 2xx audit: 2xx (graceful)
graceful	non-zero	Yes/No	In*	non-audit: 2xx audit: 2xx (graceful)
forced	N/A	Yes/No	Out	non-audit: 501 audit: 2xx (forced)
restart	zero	No	In	non-audit: 2xx,405* audit: 2xx (restart)
restart	zero	Yes	In	non-audit: 2xx audit: 2xx (restart)
restart	non-zero	No	Out*	non-audit: 501* audit: 2xx (restart)
restart	non-zero	Yes	Out*	non-audit: 501* audit: 2xx (restart)
disconnected	zero/non-zero	No	In	non-audit: 2xx, audit: 2xx (disconnected)
disconnected	zero/non-zero	Yes	In	non-audit: 2xx audit: 2xx (restart)
cancel-graceful	N/A	Yes/No	In	non-audit: 2xx audit: 2xx (restart)

Notes (*):

* The three service-states marked with "*" will change after the expiration of the RestartDelay at which time an updated RestartInProgress command SHOULD be sent.

* If the endpoint returns 2xx when the restart procedure has not yet completed, then in-order delivery MUST still be satisfied, i.e., piggy-backing is to be used. If instead, the command is not processed, 405 SHOULD be returned.

* Following a "restart" RestartInProgress with a non-zero RestartDelay, error code 501 is only returned until the endpoint goes in-service, i.e., until the expiration of the RestartDelay.

4.4.6 Fighting the Restart Avalanche

Let's suppose that a large number of gateways are powered on simultaneously. If they were to all initiate a RestartInProgress transaction, the Call Agent 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 behavior is REQUIRED:

1) When a gateway is powered on, it MUST 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 gateways that would use the same algorithm.

2) The gateway MUST then wait for either the end of this timer, the reception of a command from the Call Agent, or the detection of a local user activity, such as for example an off-hook transition on a residential gateway.

3) When the timer elapses, when a command is received, or when an activity is detected, the gateway MUST initiate the restart procedure.

The restart procedure simply requires the endpoint to guarantee that the first

* non-audit command, or

* non-restart response (i.e., error codes other than 405, 501, and 520) to a non-audit command

that the Call Agent sees from this endpoint is a "restart" RestartInProgress command. The endpoint is free to take full advantage of piggybacking to achieve this. Endpoints that are considered in-service will have a RestartMethod of "restart", whereas endpoints considered out-of-service will have a RestartMethod of "forced" (also see Section 4.4.5). Commands rejected due to an endpoint not yet having completed the restart procedure SHOULD use error code 405 (endpoint "restarting").

The restart procedure is complete once a success response has been received. If an error response is received, the subsequent behavior depends on the error code in question:

* If the error code indicates a transient error (4xx), then the restart procedure MUST be initiated again (as a new transaction).

* If the error code is 521, then the endpoint is redirected, and the restart procedure MUST be initiated again (as a new transaction). The 521 response MUST have included a NotifiedEntity which then is the "notified entity" towards which the restart is initiated. If it did not include a NotifiedEntity, the response is treated as any other permanent error (see below).

* If the error is any other permanent error (5xx), and the endpoint is not able to rectify the error, then the endpoint no longer initiates the restart procedure on its own (until rebooted/restarted) unless otherwise specified. If a command is received for the endpoint, the endpoint MUST initiate the restart procedure again.

Note that if the RestartInProgress is piggybacked with the response (R) to a command received while restarting, then retransmission of the RestartInProgress does not require piggybacking of the response R. However, while the endpoint is restarting, a resend of the response R does require the RestartInProgress to be piggybacked to ensure in-order delivery of the two.

Should the gateway enter the "disconnected" state while carrying out the restart procedure, the disconnected procedure specified in Section 4.4.7 MUST be carried out, except that a "restart" rather than "disconnected" message is sent during the procedure.

Each endpoint in a gateway will have a provisionable Call Agent, i.e., "notified entity", to direct the initial restart message towards. When the collection of endpoints in a gateway is managed by more than one Call Agent, the above procedure MUST be performed for each collection of endpoints managed by a given Call Agent. The gateway MUST take full advantage of wild-carding to minimize the

number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent. Note that during startup, it is possible for endpoints to start out as being out-of-service, and then become in-service as part of the gateway initialization procedure. A gateway may thus choose to send first a "forced" RestartInProgress for all its endpoints, and subsequently a "restart" RestartInProgress for the endpoints that come in-service. Alternatively, the gateway may simply send "restart" RestartInProgress for only those endpoints that are in-service, and "forced" RestartInProgress for the specific endpoints that are out-of-service. Wild-carding MUST still be used to minimize the number of messages sent though.

The value of MWD is a configuration parameter that depends on the type of the gateway. The following reasoning can be used to determine the value of this delay on residential gateways.

Call agents 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 MGCP transactions between each endpoint and the Call Agent. This simple calculation shows that the Call Agent is expected to handle 5 to 6 transactions for each endpoint, every 30 minutes on average, or, to put it otherwise, about one transaction per endpoint every 5 to 6 minutes on average. This suggest 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 endpoints, and also because the usage rate of these endpoints is much higher than 10% during the peak busy hour, a typical value being 60%. These endpoints, during the peak hour, are thus expected to contribute about one transaction per minute to the Call Agent load. A reasonable algorithm is to make the value of MWD per "trunk" endpoint six times shorter than the MWD per residential gateway, and also inversely proportional to the number of endpoints 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.

4.4.7 Disconnected Endpoints

In addition to the restart procedure, gateways also have a "disconnected" procedure, which MUST be initiated when an endpoint becomes "disconnected" as described in Section 4.3. It should here be noted, that endpoints can only become disconnected when they attempt to communicate with the Call Agent. The following steps MUST be followed by an endpoint that becomes "disconnected":

1. A "disconnected" timer is initialized to a random value, uniformly distributed between 1 and a provisionable "disconnected" initial waiting delay (Tdinit), e.g., 15 seconds. Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways and endpoints that would use the same algorithm.

2. The gateway then waits for either the end of this timer, the reception of a command for the endpoint from the Call Agent, or the detection of a local user activity for the endpoint, such as for example an off-hook transition.

3. When the "disconnected" timer elapses for the endpoint, when a command is received for the endpoint, or when local user activity is detected for the endpoint, the gateway initiates the "disconnected" procedure for the endpoint - if a disconnected procedure was already in progress for the endpoint, it is simply replaced by the new one. Furthermore, in the case of local user activity, a provisionable "disconnected" minimum waiting delay (Tdmin) MUST have elapsed since the endpoint became disconnected or the last time it ended the "disconnected" procedure in order to limit the rate at which the procedure is performed. If Tdmin has not passed, the endpoint simply proceeds to step 2 again, without affecting any disconnected procedure already in progress.

4. If the "disconnected" procedure still left the endpoint disconnected, the "disconnected" timer is then doubled, subject to a provisionable "disconnected" maximum waiting delay (Tdmax), e.g., 600 seconds, and the gateway proceeds with step 2 again (using a new transaction-id).

The "disconnected" procedure is similar to the restart procedure in that it simply states that the endpoint MUST send a RestartInProgress command to the Call Agent informing it that the endpoint was disconnected. Furthermore, the endpoint MUST guarantee that the first non-audit message (non-audit command or response to non-audit command) that the Call Agent sees from this endpoint MUST inform the Call Agent that the endpoint is disconnected (unless the endpoint goes out-of-service). When a command (C) is received, this is achieved by sending a piggy-backed datagram with a "disconnected"

RestartInProgress command and the response to command C to the source address of command C as opposed to the current "notified entity". This piggy-backed RestartInProgress is not automatically retransmitted by the endpoint but simply relies on fate-sharing with the piggy-backed response to guarantee the in-order delivery requirement. The Call Agent still sends a response to the piggy-backed RestartInProgress, however, as usual, the response may be lost. In addition to the piggy-backed RestartInProgress command, a new "disconnected" procedure is triggered by the command received. This will lead to a non piggy-backed copy (i.e., same transaction) of the "disconnected" RestartInProgress command being sent reliably to the current "notified entity".

When the Call Agent learns that the endpoint is disconnected, the Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint. Note that each such "disconnected" procedure will result in a new RestartInProgress command, which will be subject to the normal retransmission procedures specified in Section 4.3. At the end of the procedure, the endpoint may thus still be "disconnected". Should the endpoint go out-of-service while being disconnected, it SHOULD send a "forced" RestartInProgress message as described in Section 2.3.12.

The disconnected procedure is complete once a success response has been received. Error responses are handled similarly to the restart procedure (Section 4.4.6). If the "disconnected" procedure is to be initiated again following an error response, the rate-limiting timer considerations specified above still apply.

Note, that if the RestartInProgress is piggybacked with the response (R) to a command received while being disconnected, then retransmission of this particular RestartInProgress does not require piggybacking of the response R. However, while the endpoint is disconnected, resending the response R does require the RestartInProgress to be piggybacked with the response to ensure the in-order delivery of the two.

If a set of disconnected endpoints have the same "notified entity", and the set of endpoints can be named with a wildcard, the gateway MAY replace the individual disconnected procedures with a suitably wildcarded disconnected procedure instead. In that case, the Restart Delay for the wildcarded "disconnected" RestartInProgress command SHALL be the Restart Delay corresponding to the oldest disconnected procedure replaced. Note that if only a subset of these endpoints subsequently have their "notified entity" changed and/or are no longer disconnected, then that wildcarded disconnected procedure can no longer be used. The remaining individual disconnected procedures MUST then be resumed again.

A disconnected endpoint may wish to send a command (besides RestartInProgress) while it is disconnected. Doing so will only succeed once the Call Agent is reachable again, which raises the question of what to do with such a command meanwhile. At one extreme, the endpoint could drop the command right away, however that would not work very well when the Call Agent was in fact available, but the endpoint had not yet completed the "disconnected" procedure (consider for example the case where a NotificationRequest was just received which immediately resulted in a Notify being generated). To prevent such scenarios, disconnected endpoints SHALL NOT blindly drop new commands to be sent for a period of T-MAX seconds after they receive a non-audit command.

One way of satisfying this requirement is to employ a temporary buffering of commands to be sent, however in doing so, the endpoint MUST ensure, that it:

* does not build up a long queue of commands to be sent,

* does not swamp the Call Agent by rapidly sending too many commands once it is connected again.

Buffering commands for T-MAX seconds and, once the endpoint is connected again, limiting the rate at which buffered commands are sent to one outstanding command per endpoint is considered acceptable (see also Section 4.4.8, especially if using wildcards). If the endpoint is not connected within T-MAX seconds, but a "disconnected" procedure is initiated within T-MAX seconds, the endpoint MAY piggyback the buffered command(s) with that RestartInProgress. Note, that once a command has been sent, regardless of whether it was buffered initially, or piggybacked earlier, retransmission of that command MUST cease T-MAX seconds after the initial send as described in Section 4.3.

This specification purposely does not specify any additional behavior for a disconnected endpoint. Vendors MAY for instance choose to provide silence, play reorder tone, or even enable a downloaded wav file to be played.

The default value for Tdinit is 15 seconds, the default value for Tdmin, is 15 seconds, and the default value for Tdmax is 600 seconds.

4.4.8 Load Control in General

The previous sections have described several MGCP mechanisms to deal with congestion and overload, namely:

* the UDP retransmission strategy which adapts to network and call agent congestion on a per endpoint basis,

* the guidelines on the ordering of commands which limit the number of commands issued in parallel,

* the restart procedure which prevents flooding in case of a restart avalanche, and

* the disconnected procedure which prevents flooding in case of a large number of disconnected endpoints.

It is however still possible for a given set of endpoints, either on the same or different gateways, to issue one or more commands at a given point in time. Although it can be argued, that Call Agents should be sized to handle one message per served endpoint at any given point in time, this may not always be the case in practice. Similarly, gateways may not be able to handle a message for all of its endpoints at any given point in time. In general, such issues can be dealt with through the use of a credit-based mechanism, or by monitoring and automatically adapting to the observed behavior. We opt for the latter approach as follows.

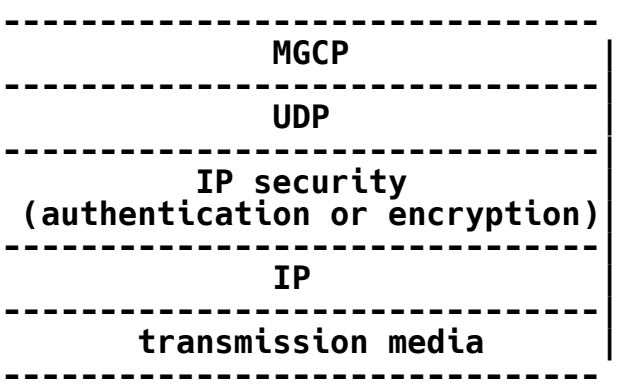
Conceptually, we assume that Call Agents and gateways maintain a queue of incoming transactions to be executed. Associated with this transaction queue is a high-water and a low-water mark. Once the queue length reaches the high-water mark, the entity SHOULD start issuing 101 provisional responses (transaction queued) until the queue length drops to the low-water mark. This applies to new transactions as well as to retransmissions. If the entity is unable to process any new transactions at this time, it SHOULD return error code 409 (processing overload).

Furthermore, gateways SHOULD adjust the sending rate of new commands to a given Call Agent by monitoring the observed response times from that Call Agent to a *set* of endpoints. If the observed smoothed average response time suddenly rises significantly over some threshold, or the gateway receives a 101 (transaction queued) or 409 (overload) response, the gateway SHOULD adjust the sending rate of new commands to that Call Agent accordingly. The details of the smoothing average algorithm, the rate adjustments, and the thresholds involved are for further study, however they MUST be configurable.

Similarly, Call Agents SHOULD adjust the sending rate of new transactions to a given gateway by monitoring the observed response times from that gateway for a *set* of endpoints. If the observed smoothed average response time suddenly rises significantly over some threshold, or the Call Agent receives a 101 (transaction queued) or 409 (overloaded), the Call Agent SHOULD adjust the sending rate of new commands to that gateway accordingly. The details of the smoothing average algorithm, the rate adjustments, and the thresholds involved are for further study, however they MUST be configurable.

5. Security Requirements

Any entity can send a command to an MGCP endpoint. If unauthorized entities could use the MGCP, they would be able to set-up unauthorized calls, or to interfere with authorized calls. We expect that MGCP messages will always be carried over secure Internet connections, as defined in the IP security architecture as defined in RFC 2401, using either the IP Authentication Header, defined in RFC 2402, or the IP Encapsulating Security Payload, defined in RFC 2406. The complete MGCP protocol stack would thus include the following layers:



Adequate protection of the connections will be achieved if the gateways and the Call Agents only accept messages for which IP security provided an authentication service. An encryption service will provide additional protection against eavesdropping, thus preventing third parties from monitoring the connections set up by a given endpoint.

The encryption service will also be requested if the session descriptions are used to carry session keys, as defined in SDP.

These procedures do not necessarily protect against denial of service attacks by misbehaving gateways or misbehaving Call Agents. However, they will provide an identification of these misbehaving entities, which should then be deprived of their authorization through maintenance procedures.

5.1 Protection of Media Connections

MGCP allows Call Agent to provide gateways 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 will be decoded and the signals will be played on the "line side".

A basic protection against this attack is to only accept packets from known sources, however this tends to conflict with RTP principles. This also has two inconveniences: it slows down connection establishment and it can be fooled by source spoofing:

* To enable the address-based protection, the Call Agent must obtain the source address of the egress gateway and pass it to the ingress gateway. 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 and 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.

6. Packages

As described in Section 2.1.6, packages are the preferred way of extending MGCP. In this section we describe the requirements associated with defining a package.

A package MUST have a unique package name defined. The package name MUST be registered with the IANA, unless it starts with the characters "x-" or "x+" which are reserved for experimental packages. Please refer to Appendix C for IANA considerations.

A package MUST also have a version defined which is simply a non-negative integer. The default and initial version of a package is zero, the next version is one, etc. New package versions MUST be completely backwards compatible, i.e., a new version of a package MUST NOT redefine or remove any of the extensions provided in an earlier version of the package. If such a need arises, a new package name MUST be used instead.

Packages containing signals of type time-out MAY indicate if the "to" parameter is supported for all the time-out signals in the package as well as the default rounding rules associated with these (see Section 3.2.2.4). If no such definition is provided, each time-out signal SHOULD provide these definitions.

A package defines one or more of the following extensions:

* Actions

* BearerInformation

* ConnectionModes

* ConnectionParameters

* DigitMapLetters

* Events and Signals

* ExtensionParameters

* LocalConnectionOptions

* ReasonCodes

* RestartMethods

* Return codes

For each of the above types of extensions supported by the package, the package definition MUST contain a description of the extension as defined in the following sections. Please note, that package extensions, just like any other extension, MUST adhere to the MGCP grammar.

6.1 Actions

Extension Actions SHALL include:

* The name and encoding of the extension action.

* If the extension action takes any action parameters, then the name, encoding, and possible values of those parameters.

* A description of the operation of the extension action.

* A listing of the actions in this specification the extension can be combined with. If such a listing is not provided, it is assumed that the extension action cannot be combined with any other action in this specification.

* If more than one extension action is defined in the package, then a listing of the actions in the package the extension can be combined with. If such a listing is not provided, it is assumed that the extension action cannot be combined with any other action in the package.

Extension actions defined in two or more different packages SHOULD NOT be used simultaneously, unless very careful consideration to their potential interaction and side-effects has been given.

6.2 BearerInformation

BearerInformation extensions SHALL include:

* The name and encoding of the BearerInformation extension.

* The possible values and encoding of those values that can be assigned to the BearerInformation extension.

* A description of the operation of the BearerInformation extension. As part of this description the default value (if any) if the extension is omitted in an EndpointConfiguration command MUST be defined. It may be necessary to make a distinction between the default value before and after the initial application of the parameter, for example if the parameter retains its previous value once specified, until explicitly altered. If default values are not described, then the extension parameter simply defaults to empty in all EndpointConfiguration commands.

Note that the extension SHALL be included in the result for an AuditEndpoint command auditing the BearerInformation.

6.3 ConnectionModes

Extension Connection Modes SHALL include:

* The name and encoding of the extension connection mode.

* A description of the operation of the extension connection mode.

* A description of the interaction a connection in the extension connection mode will have with other connections in each of the modes defined in this specification. If such a description is not provided, the extension connection mode MUST NOT have any interaction with other connections on the endpoint.

Extension connection modes SHALL NOT be included in the list of modes in a response to an AuditEndpoint for Capabilities, since the package will be reported in the list of packages.

6.4 ConnectionParameters

Extension Connection Parameters SHALL include:

* The name and encoding of the connection parameter extension.

* The possible values and encoding of those values that can be assigned to the connection parameter extension.

* A description of how those values are derived.

Note that the extension connection parameter MUST be included in the result for an AuditConnection command auditing the connection parameters.

6.5 DigitMapLetters

Extension Digit Map Letters SHALL include:

* The name and encoding of the extension digit map letter(s).

* A description of the meaning of the extension digit map letter(s).

Note that extension DigitMapLetters in a digit map do not follow the normal naming conventions for extensions defined in packages. More specifically the package name and slash ("/") will not be part of the extension name, thereby forming a flat and limited name space with potential name clashing.

Therefore, a package SHALL NOT define a digit map letter extension whose encoding has already been used in another package. If two packages have used the same encoding for a digit map letter extension, and those two packages are supported by the same endpoint, the result of using that digit map letter extension is undefined.

Note that although an extension DigitMapLetter does not include the package name prefix and slash ("/") as part of the extension name within a digit map, the package name prefix and slash are included when the event code for the event that matched the DigitMapLetter is reported as an observed event. In other words, the digit map just define the matching rule(s), but the event is still reported like any other event.

6.6 Events and Signals

The event/signal definition SHALL include the precise name of the event/signal (i.e., the code used in MGCP), a plain text definition of the event/signal, and, when appropriate, the precise definition of the corresponding events/signals, for example the exact frequencies of audio signals such as dial tones or DTMF tones.

The package description MUST provide, for each event/signal, the following information:

* The description of the event/signal and its purpose, which SHOULD include the actual signal that is generated by the client (e.g., xx ms FSK tone) as well as the resulting user observed result (e.g., Message Waiting light on/off).

The event code used for the event/signal.

* The detailed characteristics of the event/signal, such as for example frequencies and amplitude of audio signals, modulations and repetitions. Such details may be country specific.

* The typical and maximum duration of the event/signal if applicable.

* If the signal or event can be applied to a connection (across a media stream), it MUST be indicated explicitly. If no such indication is provided, it is assumed that the signal or event cannot be applied to a connection.

For events, the following MUST be provided as well:

* An indication if the event is persistent. By default, events are not persistent - defining events as being persistent is discouraged (see Appendix B for a preferred alternative). Note that persistent

events will automatically trigger a Notify when they occur, unless the Call Agent explicitly instructed the endpoint otherwise. This not only violates the normal MGCP model, but also assumes the Call Agent supports the package in question. Such an assumption is unlikely to hold in general.

* An indication if there is an auditable event-state associated with the event. By default, events do not have auditable event-states.

* If event parameters are supported, it MUST be stated explicitly. The precise syntax and semantics of these MUST then be provided (subject to the grammar provided in Appendix A). It SHOULD also be specified whether these parameters apply to RequestedEvents, ObservedEvents, DetectEvents and EventStates. If not specified otherwise, it is assumed that:

  * they do not apply to RequestedEvents,

  * they do apply to ObservedEvents,

  * they apply in the same way to DetectEvents as they do to RequestedEvents for a given event parameter,

  * they apply in the same way to EventStates as they do to ObservedEvents for a given event parameter.

* If the event is expected to be used in digit map matching, it SHOULD explicitly state so. Note that only events with single letter or digit parameter codes can do this. See Section 2.1.5 for further details.

For signals, the following MUST be provided as well:

* The type of signal (OO, TO, BR).

* Time-Out signals SHOULD have an indication of the default time-out value. In some cases, time-out values may be variable (if dependent on some action to complete such as out-pulsing digits).

* If signal parameters are supported, it MUST be stated explicitly. The precise syntax and semantics of these MUST then be provided (subject to the grammar provided in Appendix A).

* Time-Out signals may also indicate whether the "to" parameter is supported or not as well as what the rounding rules associated with them are. If omitted from the signal definition, the package-wide definition is assumed (see Section 6). If the package definition did not specify this, rounding rules default to the nearest non-

zero second, whereas support for the "to" parameter defaults to "no" for package version zero, and "yes" for package versions one and higher.

The following format is RECOMMENDED for defining events and signals in conformance with the above:

Symbol	Definition	R	S Duration

where:

* Symbol indicates the event code used for the event/signal, e.g., "hd".

* Definition gives a brief definition of the event/signal

* R contains an "x" if the event can be detected or one or more of the following symbols:

  - "P" if the event is persistent.

  - "S" if the events is an event-state that may be audited.

  - "C" if the event can be detected on a connection.

* S contains one of the following if it is a signal:

  - "OO" if the signal is On/Off signal.

  - "TO" if the signal is a Time-Out signal.

  - "BR" if the signal is a Brief signal.

* S also contains:

  - "C" if the signal can be applied on a connection.

The table SHOULD then be followed by a more comprehensive description of each event/signal defined.

6.6.1 Default and Reserved Events

All packages that contain Time-Out type signals contain the operation failure ("of") and operation complete ("oc") events, irrespective of whether they are provided as part of the package description or not. These events are needed to support Time-Out signals and cannot be overridden in packages with Time-Out signals. They MAY be extended if necessary, however such practice is discouraged.

If a package without Time-Out signals does contain definitions for the "oc" and "of" events, the event definitions provided in the package MAY over-ride those indicated here. Such practice is however discouraged and is purely allowed to avoid potential backwards compatibility problems.

It is considered good practice to explicitly mention that the two events are supported in accordance with their default definitions, which are as follows:

Symbol	Definition	R	S Duration
oc	Operation Complete	x	
of	Operation Failure	x	

Operation complete (oc): The operation complete event is generated when the gateway was asked to apply one or several signals of type TO on the endpoint or connection, and one or more of those signals completed without being stopped by the detection of a requested event such as off-hook transition or dialed digit. The completion report should carry as a parameter the name of the signal that came to the end of its live time, as in:

```
O: G/oc(G/rt)
```

In this case, the observed event occurred because the "rt" signal in the "G" package timed out.

If the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: G/oc(G/rt@0A3F58)
```

When the operation complete event is requested, it cannot be parameterized with any event parameters. When the package name is omitted (which is discouraged) as part of the signal name, the default package is assumed.

Operation failure (of): The operation failure event is generated when the endpoint was asked to apply one or several signals of type TO on the endpoint or connection, and one or more of those signals failed prior to timing out. The completion report should carry as a parameter the name of the signal that failed, as in:

O: G/of(G/rt)

In this case a failure occurred in producing the "rt" signal in the "G" package.

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

O: G/of(G/rt@0A3F58)

When the operation failure event is requested, event parameters can not be specified. When the package name is omitted (which is discouraged), the default package name is assumed.

6.7 ExtensionParameters

Extension parameter extensions SHALL include:

* The name and encoding of the extension parameter.

* The possible values and encoding of those values that can be assigned to the extension parameter.

* For each of the commands defined in this specification, whether the extension parameter is Mandatory, Optional, or Forbidden in requests as well as responses. Note that extension parameters SHOULD NOT normally be mandatory.

* A description of the operation of the extension parameter. As part of this description the default value (if any) if the extension is omitted in a command MUST be defined. It may be necessary to make a distinction between the default value before and after the initial application of the parameter, for example if the parameter retains its previous value once specified, until explicitly altered. If default values are not described, then the extension parameter simply defaults to empty in all commands.

* Whether the extension can be audited in AuditEndpoint and/or AuditConnection as well as the values returned. If nothing is specified, then auditing of the extension parameter can only be done for AuditEndpoint, and the value returned SHALL be the current value for the extension. Note that this may be empty.

6.8 LocalConnectionOptions

LocalConnectionOptions extensions SHALL include:

* The name and encoding of the LocalConnectionOptions extension.

* The possible values and encoding of those values that can be assigned to the LocalConnectionOptions extension.

* A description of the operation of the LocalConnectionOptions extension. As part of this description the following MUST be specified:

  - The default value (if any) if the extension is omitted in a CreateConnection command.

  - The default value if omitted in a ModifyConnection command. This may be to simply retain the previous value (if any) or to apply the default value. If nothing is specified, the current value is retained if possible.

  - If Auditing of capabilities will result in the extension being returned, then a description to that effect as well as with what possible values and their encoding (note that the package itself will always be returned). If nothing is specified, the extension SHALL NOT be returned when auditing capabilities.

Also note, that the extension MUST be included in the result for an AuditConnection command auditing the LocalConnectionOptions.

6.9 Reason Codes

Extension reason codes SHALL include:

* The number for the reason code. The number MUST be in the range 800 to 899.

* A description of the extension reason code including the circumstances that leads to the generation of the reason code. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension reason code correctly.

Note that the extension reason code may have to be provided in the result for an AuditEndpoint command auditing the reason code.

6.10 RestartMethods

Extension Restart Methods SHALL include:

* The name and encoding for the restart method.

* A description of the restart method including the circumstances that leads to the generation of the restart method. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension restart method correctly.

* An indication of whether the RestartDelay parameter is to be used with the extension. If nothing is specified, it is assumed that it is not to be used. In that case, RestartDelay MUST be ignored if present.

* If the restart method defines a service state, the description MUST explicitly state and describe this. In that case, the extension restart method can then be provided in the result for an AuditEndpoint command auditing the restart method.

6.11 Return Codes

Extension Return Codes SHALL include:

* The number for the extension return code. The number MUST be in the range 800 to 899.

* A description of the extension return code including the circumstances that leads to the generation of the extension return code. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension return code correctly.

7. Versions and Compatibility

7.1 Changes from RFC 2705

RFC 2705 was issued in October 1999, as the last update of draft version 0.5. This updated document benefits from further implementation experience. The main changes from RFC 2705 are:

* Contains several clarifications, editorial changes and resolution of known inconsistencies.

* Firmed up specification language in accordance with RFC 2119 and added RFC 2119 conventions section.

* Clarified behavior of mixed wild-carding in endpoint names.

* Deleted naming requirement about having first term identify the physical gateway when the gateway consists of multiple physical gateways. Also added recommendations on wild-carding naming usage from the right only, as well as mixed wildcard usage.

* Clarified that synonymous forms and values for endpoint names are not freely interchangeable.

* Allowed IPv6 addresses in endpoint names.

* Clarified Digit Map matching rules.

* Added missing semantics for symbols used in digit maps.

* Added Timer T description in Digit Maps.

* Added recommendation to support digit map sizes of at least 2048 bytes per endpoint.

* Clarified use of wildcards in several commands.

* Event and Signal Parameters formally defined for events and signals.

* Persistent events now allowed in base MGCP protocol.

* Added additional detail on connection wildcards.

* Clarified behavior of loopback, and continuity test connection modes for mixing and multiple connections in those modes.

* Modified BearerInformation to be conditional optional in the EndpointConfiguration command.

* Clarified "swap audio" action operation for one specific scenario and noted that operation for other scenarios is undefined.

* Added recommendation that all implementations support PCMU encoding for interoperability.

* Changed Bandwidth LocalConnectionOptions value from excluding to including overhead from the IP layer and up for consistency with SDP.

* Clarified that mode of second connection in a CreateConnection command will be set to "send/receive".

* Type of service default changed to zero.

* Additional detail on echo cancellation, silence suppression, and gain control. Also added recommendation for Call Agents not to specify handling of echo cancellation and gain control.

* Added requirement for a connection to have a RemoteConnectionDescriptor in order to use the "network loopback" and "network continuity test" modes.

* Removed procedures and specification for NAS's (will be provided as package instead).

* Removed procedures and specification for ATM (will be provided as package instead).

* Added missing optional NotifiedEntity parameter to the DeleteConnection (from the Call Agent) MGCI command.

* Added optional new MaxMGCPDatagram RequestedInfo code for AuditEndpoint to enable auditing of maximum size of MGCP datagrams supported.

* Added optional new PackageList RequestedInfo code for AuditEndpoint to enable auditing of packages with a package version number. PackageList parameter also allowed with return code 518 (unsupported package).

* Added missing attributes in Capabilities.

* Clarified that at the expiration of a non-zero restart delay, an updated RestartInProgress should be sent. Also clarified that a new NotifiedEntity can only be returned in response to a RestartInProgress command.

* Added Response Acknowledgement response (return code 000) and included in three-way handshake.

* ResponseAck parameter changed to be allowed in all commands.

* Added return codes 101, 405, 406, 407, 409, 410, 503, 504, 505, 506, 507, 508, 509, 533, 534, 535, 536, 537, 538, 539, 540, 541, and defined return codes in range 800-899 to be package specific return codes. Additional text provided for some return codes and additional detail on how to handle unknown return codes added.

* Added reason code 903, 904, 905 and defined reason codes 800-899 to be package specific reason codes.

* Added section clarifying codec negotiation procedure.

* Clarified that resource reservation parameters in a ModifyConnection command defaults to the current value used.

* Clarified that connection mode is optional in ModifyConnection commands.

* Corrected LocalConnectionDescriptor to be optional in response to CreateConnection commands (in case of failure).

* Clarified that quoted-strings are UTF-8 encoded and interchangeability of quoted strings and unquoted strings.

* Clarified that Transaction Identifiers are compared as numerical values.

* Clarified bit-ordering for Type Of Service LocalConnectionOptions.

* Clarified the use of RequestIdentifier zero.

* Added example sections for commands, responses, and some call flows.

* Corrected usage of and requirements for SDP to be strictly RFC 2327 compliant.

* Added requirement that all MGCP implementations must support MGCP datagrams up to at least 4000 bytes. Also added new section on Maximum Datagram Size, Fragmentation and reassembly.

* Generalized piggybacking retransmission scheme to only state underlying requirements to be satisfied.

* Clarified the section on computing retransmission timers.

* Clarified operation of long-running transactions, including provisional responses, retransmissions and failures.

* Enhanced description of provisional responses and interaction with three-way handshake.

* Enhanced description of fail-over and the role of "notified entity". An empty "notified entity" has been allowed, although strongly discouraged.

* Clarified retransmission procedure and removed "wrong key" considerations from it. Also fixed inconsistencies between Max1 and Max2 retransmission boundaries and the associated flow diagram.

* Updated domain name resolution for retransmission procedure to incur less overhead when multiple endpoints are retransmitting.

* Removed requirement for in-order delivery of NotificationRequests response and Notify commands. Notify commands are still delivered in-order.

* Clarified that activating an embedded Notification Request does not clear the list of ObservedEvents.

* Defined interactions between disconnected state and notification state.

* Added section on transactional semantics.

* Defined gateway behavior when multiple interacting transactions are received.

* Additional details provided on service states. Clarified relationship between endpoint service states, restart methods, and associated processing of commands.

* Clarified operation for transitioning from "restart procedure" to "disconnected state".

* Allowed auditing commands and responses to bypass the "restart" and "disconnected" procedures.

* Clarified operation of "disconnected procedure" and in particular the operation of piggy-backed "disconnected" RestartInProgress messages.

* Added option to aggregate "disconnected" RestartInProgress messages under certain conditions to reduce message volume.

* Defined additional behavior for endpoints wishing to send commands while in the "disconnected" state.

* Added new section on Load Control in General which includes two new error codes (101 and 409) to handle overload.

* Deleted the "Proposed MoveConnection command".

* Removed packages from protocol specification (will be provided in separate documents instead).

* Package concept formally extended to be primary extension mechanism now allowing extensions for the following to be defined in packages as well:

  - BearerInformation

  - LocalConnectionOptions

  - ExtensionParameters

  - Connection Modes

  - Actions

  - Digit Map Letters

  - Connection Parameters

  - Restart Methods

  - Reason Codes

  - Return Codes

* Requirements and suggested format for package definitions added.

* Defined "operation complete" and "operation failure" events to be automatically present in packages with Time-Out signals.

* Deleted list of differences that were prior to RFC 2705.

* Added Base Package to deal with quarantine buffer overflow, ObservedEvents overflow, embedded NotificationRequest failure, and to enable events to be requested persistently. A new "Message" command is included as well.

* IANA registration procedures for packages and other extensions added.

* Updated grammar to fix known errors and support new extensions in a backwards compatible manner. Added new (optional) PackageList and MaxMGCPDatagram for auditing. Changed explicit white space rules in some productions to make grammar more consistent.

* Connection Mode interaction table added.

* Added additional detail on virtual endpoint naming conventions. Also added suggested gateway endpoint convention and a "Range Wildcard" option to the Endpoint Naming Conventions.

8. Security Considerations

Security issues are discussed in section 5.

9. Acknowledgements

Special thanks are due to the authors of the original MGCP 1.0 specification: Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema, and Scott Picket.

We also want to thank the many reviewers who provided advice on the design of SGCP and then MGCP, notably Sankar Ardhanari, Francois Berard, David Auerbach, Bob Biskner, David Bukovinsky, Charles Eckel, Mario Edini, Ed Guy, Barry Hoffner, Jerry Kamitses, Oren Kudevitzki, Rajesh Kumar, Troy Morley, Dave Oran, Jeff Orwick, John Pickens, Lou Rubin, Chip Sharp, Paul Sijben, Kurt Steinbrenner, Joe Stone, and Stuart Wray.

The version 0.1 of MGCP was heavily inspired by the "Internet Protocol Device Control" (IPDC) designed by the Technical Advisory Committee set up by Level 3 Communications. Whole sets of text were retrieved from the IP Connection Control protocol, IP Media Control protocol, and IP Device Management. The authors wish to acknowledge the contribution to these protocols made by Ilya Akramovich, Bob Bell, Dan Brendes, Peter Chung, John Clark, Russ Dehlinger, Andrew Dugan, Isaac Elliott, Cary FitzGerald, Jan Gronski, Tom Hess, Geoff Jordan, Tony Lam, Shawn Lewis, Dave Mazik, Alan Mikhak, Pete O'Connell, Scott Pickett, Shyamal Prasad, Eric Presworsky, Paul Richards, Dale Skran, Louise Spergel, David Sprague, Raj Srinivasan, Tom Taylor and Michael Thomas.

10. References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[4] Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

[5] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[6] Handley, M., Perkins, C. and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[7] Rosenberg, J., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schulzrinne, H. and E. Schooler, "Session Initiation Protocol (SIP)", RFC 3261, June 2002.

[8] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[9] ITU-T, Recommendation Q.761, "FUNCTIONAL DESCRIPTION OF THE ISDN USER PART OF SIGNALING SYSTEM No. 7", (Malaga-Torremolinos, 1984; modified at Helsinki, 1993).

[10] ITU-T, Recommendation Q.762, "GENERAL FUNCTION OF MESSAGES AND SIGNALS OF THE ISDN USER PART OF SIGNALING SYSTEM No. 7", (MalagaTorremolinos, 1984; modified at Helsinki, 1993).

[11] ITU-T, Recommendation H.323 (02/98), "PACKET-BASED MULTIMEDIA COMMUNICATIONS SYSTEMS".

[12] ITU-T, Recommendation H.225, "Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems".

[13] ITU-T, Recommendation H.245 (02/98), "CONTROL PROTOCOL FOR MULTIMEDIA COMMUNICATION".

[14] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[15] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[16] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[17] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[18] Stevens, W. Richard, "TCP/IP Illustrated, Volume 1, The Protocols", Addison-Wesley, 1994.

[19] Allman, M., Paxson, V. "On Estimating End-to-End Network Path Properties", Proc. SIGCOMM'99, 1999.

[20] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[21] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989.

[22] Bellcore, "LSSGR: Switching System Generic Requirements for Call Control Using the Integrated Services Digital Network User Part (ISDNUP)", GR-317-CORE, Issue 2, December 1997.

[23] Narten, T., and Alvestrand H., "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, October 1998.

Appendix A: Formal Syntax Description of the Protocol

In this section, we provide a formal description of the protocol syntax, following the "Augmented BNF for Syntax Specifications" defined in RFC 2234. The syntax makes use of the core rules defined in RFC 2234, Section 6.1, which are not included here. Furthermore, the syntax follows the case-sensitivity rules of RFC 2234, i.e., MGCP is case-insensitive (but SDP is not). It should be noted, that ABNF does not provide for implicit specification of linear white space and MGCP messages MUST thus follow the explicit linear white space rules provided in the grammar below. However, in line with general robustness principles, implementers are strongly encouraged to tolerate additional linear white space in messages received.

```
MGCPMessage = MGCPCommand / MGCPResponse

MGCPCommand = MGCPCommandLine 0*(MGCPParameter) [EOL *SDPinformation]

MGCPCommandLine = MGCPVerb 1*(WSP) transaction-id 1*(WSP)
                        endpointName 1*(WSP) MGCPversion EOL

MGCPVerb = "EPCF" / "CRCX" / "MDCX" / "DLCX" / "RQNT"
         / "NTFY" / "AUEP" / "AUCX" / "RSIP" / extensionVerb

extensionVerb  = ALPHA 3(ALPHA / DIGIT) ; experimental starts with X

transaction-id = 1*9(DIGIT)

endpointName      = LocalEndpointName "@" DomainName
LocalEndpointName = LocalNamePart 0*("/" LocalNamePart)
LocalNamePart     = AnyName / AllName / NameString
AnyName           = "$"
AllName           = "*"
NameString        = 1*(range-of-allowed-characters)
; VCHAR except "$", "*", "/", "@"
range-of-allowed-characters  = %x21-23 / %x25-29 / %x2B-2E
                             / %x30-3F / %x41-7E

DomainName = 1*255(ALPHA / DIGIT / "." / "-")    ; as defined
           / "#" number                          ; in RFC 821
           / "[" IPv4address / IPv6address "]"   ; see RFC 2373

; Rewritten to ABNF from RFC 821
number =  1*DIGIT

;From RFC 2373
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
```

```
; this production, while occurring in RFC2373, is not referenced
; IPv6prefix  = hexpart "/" 1*2DIGIT
hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq  = hex4 *( ":" hex4)
hex4    = 1*4HEXDIG

MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
                            [1*(WSP) ProfileName]
ProfileName = VCHAR *( WSP / VCHAR)

MGCPParameter = ParameterValue EOL

; Check infoCode if more parameter values defined
; Most optional values can only be omitted when auditing
ParameterValue = ("K"  ":" 0*(WSP)  [ResponseAck])
               / ("B"  ":" 0*(WSP)  [BearerInformation])
               / ("C"  ":" 0*(WSP)  CallId)
               / ("I"  ":" 0*(WSP)  [ConnectionId])
               / ("N"  ":" 0*(WSP)  [NotifiedEntity])
               / ("X"  ":" 0*(WSP)  [RequestIdentifier])
               / ("L"  ":" 0*(WSP)  [LocalConnectionOptions])
               / ("M"  ":" 0*(WSP)  ConnectionMode)
               / ("R"  ":" 0*(WSP)  [RequestedEvents])
               / ("S"  ":" 0*(WSP)  [SignalRequests])
               / ("D"  ":" 0*(WSP)  [DigitMap])
               / ("O"  ":" 0*(WSP)  [ObservedEvents])
               / ("P"  ":" 0*(WSP)  [ConnectionParameters])
               / ("E"  ":" 0*(WSP)  ReasonCode)
               / ("Z"  ":" 0*(WSP)  [SpecificEndpointID])
               / ("Z2" ":" 0*(WSP)  SecondEndpointID)
               / ("I2" ":" 0*(WSP)  SecondConnectionID)
               / ("F"  ":" 0*(WSP)  [RequestedInfo])
               / ("Q"  ":" 0*(WSP)  QuarantineHandling)
               / ("T"  ":" 0*(WSP)  [DetectEvents])
               / ("RM" ":" 0*(WSP)  RestartMethod)
               / ("RD" ":" 0*(WSP)  RestartDelay)
               / ("A"  ":" 0*(WSP)  [Capabilities])
               / ("ES" ":" 0*(WSP)  [EventStates])
               / ("PL" ":" 0*(WSP)  [PackageList])    ; Auditing only
               / ("MD" ":" 0*(WSP)  MaxMGCPDatagram)  ; Auditing only
               / (extensionParameter ":" 0*(WSP) [parameterString])

; A final response may include an empty ResponseAck
ResponseAck =  confirmedTransactionIdRange
               *( ","  0*(WSP) confirmedTransactionIdRange )

confirmedTransactionIdRange = transaction-id ["-" transaction-id]
```

```
BearerInformation = BearerAttribute 0*("," 0*(WSP) BearerAttribute)
BearerAttribute   = ("e" ":" BearerEncoding)
                  / (BearerExtensionName [":" BearerExtensionValue])
BearerExtensionName  = PackageLCOExtensionName
BearerExtensionValue = LocalOptionExtensionValue
BearerEncoding = "A" / "mu"

CallId = 1*32(HEXDIG)

; The audit request response may include a list of identifiers
ConnectionId = 1*32(HEXDIG) 0*("," 0*(WSP) 1*32(HEXDIG))
SecondConnectionID = ConnectionId

NotifiedEntity = [LocalName "@"] DomainName [":" portNumber]
LocalName  = LocalEndpointName            ; No internal structure

portNumber = 1*5(DIGIT)

RequestIdentifier = 1*32(HEXDIG)

LocalConnectionOptions = LocalOptionValue 0*(WSP)
                         0*("," 0*(WSP) LocalOptionValue 0*(WSP))
LocalOptionValue = ("p"  ":" packetizationPeriod)
                 / ("a"  ":" compressionAlgorithm)
                 / ("b"  ":" bandwidth)
                 / ("e"  ":" echoCancellation)
                 / ("gc" ":" gainControl)
                 / ("s"  ":" silenceSuppression)
                 / ("t"  ":" typeOfService)
                 / ("r"  ":" resourceReservation)
                 / ("k"  ":" encryptiondata)
                 / ("nt" ":" ( typeOfNetwork /
                                supportedTypeOfNetwork))
                 / (LocalOptionExtensionName
                        [":" LocalOptionExtensionValue])

Capabilities    =  CapabilityValue 0*(WSP)
                   0*("," 0*(WSP) CapabilityValue 0*(WSP))
CapabilityValue = LocalOptionValue
                / ("v" ":" supportedPackages)
                / ("m" ":" supportedModes)

PackageList     = pkgNameAndVers 0*("," pkgNameAndVers)
pkgNameAndVers  = packageName ":" packageVersion
packageVersion  = 1*(DIGIT)

packetizationPeriod  = 1*4(DIGIT) ["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
```

```
algorithmName        = 1*(SuitableLCOCharacter)
bandwidth            = 1*4(DIGIT) ["-" 1*4(DIGIT)]
echoCancellation     = "on" / "off"
gainControl          = "auto" / ["-"] 1*4(DIGIT)
silenceSuppression   = "on" / "off"
typeOfService        = 1*2(HEXDIG)     ; 1 hex only for capabilities
resourceReservation  = "g" / "cl" / "be"

;encryption parameters are coded as in SDP (RFC 2327)
;NOTE: encryption key may contain an algorithm as specified in RFC 1890
encryptiondata = ( "clear" ":" encryptionKey )
               / ( "base64" ":" encodedEncryptionKey )
               / ( "uri" ":" URItoObtainKey )
               / ( "prompt" ) ; defined in SDP, not usable in MGCP!

encryptionKey = 1*(SuitableLCOCharacter) / quotedString
; See RFC 2045
encodedEncryptionKey = 1*(ALPHA / DIGIT / "+" / "/" / "=")
URItoObtainKey = 1*(SuitableLCOCharacter) / quotedString

typeOfNetwork = "IN" / "ATM" / "LOCAL" / OtherTypeOfNetwork
; Registered with IANA - see RFC 2327
OtherTypeOfNetwork     = 1*(SuitableLCOCharacter)
supportedTypeOfNetwork = typeOfNetwork *(";" typeOfNetwork)

supportedModes    = ConnectionMode 0*(";" ConnectionMode)

supportedPackages = packageName 0*(";" packageName)

packageName = 1*(ALPHA / DIGIT / HYPHEN) ; Hyphen neither first or last

LocalOptionExtensionName = VendorLCOExtensionName
                         / PackageLCOExtensionName
                         / OtherLCOExtensionName
VendorLCOExtensionName   = "x" ("+"/"-") 1*32(SuitableExtLCOCharacter)
PackageLCOExtensionName  = packageName "/"
                           1*32(SuitablePkgExtLCOCharacter)
; must not start with "x-" or "x+"
OtherLCOExtensionName    = 1*32(SuitableExtLCOCharacter)

LocalOptionExtensionValue = (1*(SuitableExtLCOValChar)
                                                / quotedString)
                            *(";" (1*(SuitableExtLCOValChar)
                                                / quotedString))

;Note: No "data" mode.
ConnectionMode = "sendonly" / "recvonly" / "sendrecv"
               / "confrnce" / "inactive" / "loopback"
```

```
               / "conttest" / "netwloop" / "netwtest"
               / ExtensionConnectionMode
ExtensionConnectionMode = PkgExtConnectionMode
PkgExtConnectionMode    = packageName "/" 1*(ALPHA / DIGIT)

RequestedEvents = requestedEvent 0*("," 0*(WSP) requestedEvent)
requestedEvent  = (eventName ["(" requestedActions ")"])
                / (eventName "(" requestedActions ")"
                                        "(" eventParameters ")" )
eventName = [(packageName / "*") "/"]
                (eventId / "all" / eventRange
                                        / "*" / "#") ; for DTMF
                                ["@" (ConnectionId / "$" / "*")]
eventId = 1*(ALPHA / DIGIT / HYPHEN)   ; Hyphen neither first nor last
eventRange = "[" 1*(DigitMapLetter / (DIGIT "-" DIGIT) /
                        (DTMFLetter "-" DTMFLetter)) "]"
DTMFLetter = "A" / "B" / "C" / "D"

requestedActions = requestedAction 0*("," 0*(WSP) requestedAction)
requestedAction  = "N" / "A" / "D" / "S" / "I" / "K"
                 / "E" "(" EmbeddedRequest ")"
                 / ExtensionAction
ExtensionAction  = PackageExtAction
PackageExtAction = packageName "/" Action ["(" ActionParameters ")"]
Action           = 1*ALPHA
ActionParameters = eventParameters        ; May contain actions

;NOTE: Should tolerate different order when receiving, e.g., for NCS.
EmbeddedRequest = (      "R" "(" EmbeddedRequestList ")"
                    ["," 0*(WSP) "S" "(" EmbeddedSignalRequest ")"]
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"]      )
                / (      "S" "(" EmbeddedSignalRequest ")"
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"] )
                / (      "D" "(" EmbeddedDigitMap ")" )

EmbeddedRequestList   = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap

SignalRequests   = SignalRequest 0*("," 0*(WSP) SignalRequest )
SignalRequest    = eventName [ "(" eventParameters ")" ]

eventParameters  = eventParameter 0*("," 0*(WSP) eventParameter)
eventParameter   = eventParameterValue
                 / eventParameterName "=" eventParameter
                 / eventParameterName "(" eventParameters ")"
eventParameterString = 1*(SuitableEventParamCharacter)
eventParameterName   = eventParameterString
```

```
eventParameterValue  = eventParameterString / quotedString

DigitMap           = DigitString  / "(" DigitStringList ")"
DigitStringList    = DigitString 0*( "|" DigitString )
DigitString        = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition      = DigitMapLetter / DigitMapRange
; NOTE "X" is now included
DigitMapLetter     = DIGIT / "#" / "*" / "A" / "B" / "C" / "D" / "T"
                   / "X" / ExtensionDigitMapLetter
ExtensionDigitMapLetter = "E" / "F" / "G" / "H" / "I" / "J" / "K"
                        / "L" / "M" / "N" / "O" / "P" / "Q" / "R"
                        / "S" / "U" / "V" / "W" / "Y" / "Z"
; NOTE "[x]" is now allowed
DigitMapRange = "[" 1*DigitLetter "]"
DigitLetter   = *((DIGIT "-" DIGIT) / DigitMapLetter)

ObservedEvents = SignalRequests

EventStates    = SignalRequests

ConnectionParameters = ConnectionParameter
                        0*( "," 0*(WSP) ConnectionParameter )

ConnectionParameter  = ( "PS" "=" packetsSent )
                     / ( "OS" "=" octetsSent )
                     / ( "PR" "=" packetsReceived )
                     / ( "OR" "=" octetsReceived )
                     / ( "PL" "=" packetsLost )
                     / ( "JI" "=" jitter )
                     / ( "LA" "=" averageLatency )
                     / ( ConnectionParameterExtensionName
                              "=" ConnectionParameterExtensionValue )
packetsSent     = 1*9(DIGIT)
octetsSent      = 1*9(DIGIT)
packetsReceived = 1*9(DIGIT)
octetsReceived  = 1*9(DIGIT)
packetsLost     = 1*9(DIGIT)
jitter          = 1*9(DIGIT)
averageLatency  = 1*9(DIGIT)

ConnectionParameterExtensionName = VendorCPExtensionName
                                 /    PackageCPExtensionName
VendorCPExtensionName  = "X" "-" 2*ALPHA
PackageCPExtensionName = packageName "/" CPName
CPName = 1*(ALPHA / DIGIT / HYPHEN)
ConnectionParameterExtensionValue = 1*9(DIGIT)
```

```
MaxMGCPDatagram = 1*9(DIGIT)

ReasonCode = 3DIGIT
             [1*(WSP) "/" packageName]     ; Only for 8xx
             [WSP 1*(%x20-7E)]

SpecificEndpointID = endpointName
SecondEndpointID   = endpointName

RequestedInfo = infoCode 0*("," 0*(WSP) infoCode)

infoCode = "B" / "C" / "I" / "N" / "X" / "L" / "M" / "R" / "S"
         / "D" / "O" / "P" / "E" / "Z" / "Q" / "T" / "RC" / "LC"
         / "A" / "ES" / "RM" / "RD" / "PL" / "MD" / extensionParameter

QuarantineHandling = loopControl / processControl
                   / (loopControl "," 0*(WSP) processControl )
loopControl    = "step" / "loop"
processControl = "process" / "discard"

DetectEvents = SignalRequests

RestartMethod = "graceful" / "forced" / "restart" / "disconnected"
              / "cancel-graceful" / extensionRestartMethod
extensionRestartMethod = PackageExtensionRM
PackageExtensionRM     = packageName "/" 1*32(ALPHA / DIGIT / HYPHEN)
RestartDelay = 1*6(DIGIT)

extensionParameter = VendorExtensionParameter
                   / PackageExtensionParameter
                   / OtherExtensionParameter
VendorExtensionParameter  = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
PackageExtensionParameter = packageName "/"
                            1*32(ALPHA / DIGIT / HYPHEN)
; must not start with "x-" or x+"
OtherExtensionParameter   = 1*32(ALPHA / DIGIT / HYPHEN)

;If first character is a double-quote, then it is a quoted-string
parameterString = (%x21 / %x23-7F) *(%x20-7F) ; first and last must not
                                              ; be white space
                    / quotedString

MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                                        *2(EOL *SDPinformation)

MGCPResponseLine = responseCode 1*(WSP) transaction-id
                        [1*(WSP) "/" packageName]    ; Only for 8xx
                             [WSP responseString] EOL
```

```
responseCode = 3DIGIT
responseString = *(%x20-7E)

SuitablePkgExtLCOCharacter =  SuitableLCOCharacter

SuitableExtLCOCharacter = DIGIT / ALPHA / "+" / "-" / "_" / "&"
              / "!" / "'" / "|" / "=" / "#" / "?"
              / "." / "$" / "*" /       "@" / "[" / "]"
              / "^" / "`" / "{" / "}" / "~"

SuitableLCOCharacter   = SuitableExtLCOCharacter / "/"

SuitableExtLCOValChar  = SuitableLCOCharacter / ":"

; VCHAR except """, "(", ")", ",", and "="
SuitableEventParamCharacter = %x21 / %x23-27 / %x2A-2B
                            / %x2D-3C / %x3E-7E

; NOTE: UTF8 encoded
quotedString  = DQUOTE 0*(quoteEscape / quoteChar) DQUOTE
quoteEscape   = DQUOTE DQUOTE
quoteChar = (%x00-21 / %x23-FF)

EOL = CRLF / LF

HYPHEN = "-"

; See RFC 2327 for proper SDP grammar instead.
SDPinformation = SDPLine CRLF *(SDPLine CRLF)        ; see RFC 2327
SDPLine        = 1*(%x01-09 / %x0B / %x0C / %x0E-FF) ; for proper def.
```

Appendix B: Base Package

Package name: B
Version: 0

The MGCP specification defines a base package which contains a set of events and extension parameters that are of general use to the protocol. Although not required, it is highly RECOMMENDED to support this package as it provides important functionality for the base protocol.

B.1 Events

The table below lists the events:

Symbol	Definition	R	S	Duration
enf(##)	embedded RQNT failure	x		
oef	observed events full	x		
qbo	quarantine buffer overflow	x		

The events are defined as follows:

Embedded NotificationRequest failure (enf):
The Embedded NotificationRequest Failure (enf) event is generated when an embedded Notification Request failure occurs. When the event is requested, it should be as part of the Embedded NotificationRequest itself. When the event is reported, it may be parameterized with an error code (see Section 2.4) detailing the error that occurred. When requested, it cannot be parameterized.

Observed events full (oef):
The event is generated when the endpoint is unable to accumulate any more events in the list of ObservedEvents. If this event occurs, and it is not used to trigger a Notify, subsequent events that should have been added to the list will be lost.

Quarantine buffer overflow (qbo):
The event is generated when the quarantine buffer overflows and one or more events have been lost.

B.2 Extension Parameters

B.2.1 PersistentEvents

PersistentEvents: A list of events that the gateway is requested to detect and report persistently. The parameter is optional but can be provided in any command where the DetectEvents parameter can be provided. The initial default value of the parameter is empty. When the parameter is omitted from a command, it retains its current value. When the parameter is provided, it completely replaces the current value. Providing an event in this list, is similar (but preferable) to defining that particular event as being persistent. The current list of PersistentEvents will implicitly apply to the current as well as subsequent NotificationRequests, however no glare detection etc. will be performed (similarly to DetectEvents). If an event provided in this list is included in a RequestedEvents list, the action and event parameters used in the RequestedEvents will replace the action and event parameters associated with the event in the PersistentEvents list for the life of the RequestedEvents list, after which the PersistentEvents action and event parameters are restored. Events with event states requested through this parameter will be included in the list of EventStates if audited.

PersistentEvents can also be used to detect events on connections. Use of the "all connections" wildcard is straightforward, whereas using PersistentEvents with one or more specific connections must be considered carefully. Once the connection in question is deleted, a subsequent NotificationRequest without a new PersistentEvents value will fail (error code 515 - incorrect connection-id, is RECOMMENDED), as it implicitly refers to the deleted connection.

The parameter generates the relevant error codes from the base protocol, e.g., error code 512 if an unknown event is specified.

The PersistentEvents parameter can be audited, in which case it will return its current value. Auditing of RequestedEvents is not affected by this extension, i.e., events specified in this list are not automatically reported when auditing RequestedEvents.

The parameter name for PersistentEvents is "PR" and it is defined by the production:

```
PersistentEvents = "PR" ":" 0*WSP  [RequestedEvents]
```

The following example illustrates the use of the parameter:

```
B/PR: L/hd(N), L/hf(N), L/hu(N), B/enf, B/oef, B/qbo
```

which instructs the endpoint to persistently detect and report off-hook, hook-flash, and on-hook. It also instructs the endpoint to persistently detect and report Embedded Notification Request failure, Observed events full, and Quarantine buffer overflow.

B.2.2 NotificationState

NotificationState is a RequestedInfo parameter that can be audited with the AuditEndpoint command. It can be used to determine if the endpoint is in the notification state or not.

The parameter is forbidden in any command. In responses, it is a valid response parameter for AuditEndpoint only.

It is defined by the following grammar:

```
NotificationState        = "NS" ":" 0*WSP NotificationStateValue
NotificationStateValue   = "ns" / "ls" / "o"
```

It is requested as part of auditing by including the parameter code in RequestedInfo, as in:

```
F: B/NS
```

The response parameter will contain the value "ns" if the endpoint is in the "notification state", the value "ls" if the endpoint is in the "lockstep state" (i.e., waiting for an RQNT after a response to a NTFY has been received when operating in "step" mode), or the value "o" otherwise, as for example:

```
B/NS: ns
```

B.3 Verbs

MGCP packages are not intended to define new commands, however an exception is made in this case in order to add an important general capability currently missing, namely the ability for the gateway to send a generic message to the Call Agent.

The definition of the new command is:

```
       ReturnCode
       <-- Message(EndpointId
                   [, ...])
```

EndpointId is the name for the endpoint(s) in the gateway which is issuing the Message command. The identifier MUST be a fully qualified endpoint identifier, including the domain name of the gateway. The local part of the endpoint name MUST NOT use the "any of" wildcard.

The only parameter specified in the definition of the Message command is the EndpointId, however, it is envisioned that extensions will define additional parameters to be used with the Message command. Such extensions MUST NOT alter or otherwise interfere with the normal operation of the basic MGCP protocol. They may however define additional capabilities above and beyond that provided by the basic MGCP protocol. For example, an extension to enable the gateway to audit the packages supported by the Call Agent could be defined, whereas using the Message command as an alternative way of reporting observed events would be illegal, as that would alter the normal MGCP protocol behavior.

In order to not interfere with normal MGCP operation, lack of a response to the Message command MUST NOT lead the endpoint to become disconnected. The endpoint(s) MUST be prepared to handle this transparently and continue normal processing unaffected.

If the endpoint(s) receive a response indicating that the Call Agent does not support the Message command, the endpoint(s) MUST NOT send a Message command again until the current "notified entity" has changed. Similarly, if the endpoint(s) receive a response indicating that the Call Agent does not support one or more parameters in the Message command, the endpoint(s) MUST NOT send a Message command with those parameters again until the current "notified entity" has changed.

The Message command is encoded as MESG, as shown in the following example:

```
MESG 1200 aaln/1@rgw.whatever.net MGCP 1.0
```

Appendix C: IANA Considerations

C.1 New MGCP Package Sub-Registry

The IANA has established a new sub-registry for MGCP packages under http://www.iana.org/assignments/mgcp-packages.

Packages can be registered with the IANA according to the following procedure:

The package MUST have a unique string name which MUST NOT start with the two characters "x-" or "x+".

The package title, name, and version (zero assumed by default) MUST be registered with IANA as well as a reference to the document that describes the package. The document MUST have a stable URL and MUST be contained on a public web server.

Packages may define one or more Extension Digit Map Letters, however these are taken from a limited and flat name space. To prevent name clashing, IANA SHALL NOT register a package that defines an Extension Digit Map Letter already defined in another package registered by IANA. To ease this task, such packages SHALL contain the line "Extension Digit Map Letters: " followed by a list of the Extension Digit Map Letters defined in the package at the beginning of the package definition.

A contact name, e-mail and postal address for the package MUST be provided. The contact information SHALL be updated by the defining organization as necessary.

Finally, prior to registering a package, the IANA MUST have a designated expert [23] review the package. The expert reviewer will send e-mail to the IANA on the overall review determination.

C.2 New MGCP Package

This document defines a new MGCP Base Package in Appendix B, which has been registered by IANA.

C.3 New MGCP LocalConnectionOptions Sub-Registry

The IANA has established a new sub-registry for MGCP LocalConnectionOptions under http://www.iana.org/assignments/mgcp-localconnectionoptions.

Packages are the preferred extension mechanism, however for backwards compatibility, local connection options beyond those provided in this specification can be registered with IANA. Each such local connection option MUST have a unique string name which MUST NOT start with "x-" or "x+". The local connection option field name and encoding name MUST be registered with IANA as well as a reference to the document that describes the local connection option. The document MUST have a stable URL and MUST be contained on a public web server.

A contact name, e-mail and postal address for the local connection option MUST be provided. The contact information SHALL be updated by the defining organization as necessary.

Finally, prior to registering a LocalConnectionOption, the IANA MUST have a designated expert [23] review the LocalConnectionOption. The expert reviewer will send e-mail to the IANA on the overall review determination.

Appendix D: Mode Interactions

An MGCP endpoint can establish one or more media streams. These streams are either incoming (from a remote endpoint) or outgoing (generated at the handset microphone). The "connection mode" parameter establishes the direction and generation of these streams. When there is only one connection to an endpoint, the mapping of these streams is straightforward; the handset plays the incoming stream over the handset speaker and generates the outgoing stream from the handset microphone signal, depending on the mode parameter.

However, when several connections are established to an endpoint, there can be many incoming and outgoing streams. Depending on the connection mode used, these streams may interact differently with each other and the streams going to/from the handset.

The table below describes how different connections SHALL be mixed when one or more connections are concurrently "active". An active connection is here defined as a connection that is in one of the following modes:

* "send/receive"
* "send only"
* "receive only"
* "conference"

Connections in "network loopback", "network continuity test", or "inactive" modes are not affected by connections in the "active" modes. The Table uses the following conventions:

* Ai is the incoming media stream from Connection A
* Bi is the incoming media stream from Connection B
* Hi is the incoming media stream from the Handset Microphone
* Ao is the outgoing media stream to Connection A
* Bo is the outgoing media stream to Connection B
* Ho is the outgoing media stream to the Handset earpiece
* NA indicates no stream whatsoever (assuming there are no signals applied on the connection)

"netw" in the following table indicates either "netwloop" or "netwtest" mode.

		Connection A Mode					
		sendonly	recvonly	sendrecv	confrnce	inactive	netw
Connection B Mode	Send only	Ao=Hi Bo=Hi Ho=NA	Ao=NA Bo=Hi Ho=Ai	Ao=Hi Bo=Hi Ho=Ai	Ao=Hi Bo=Hi Ho=Ai	Ao=NA Bo=Hi Ho=NA	Ao=Ai Bo=Hi Ho=NA
	recv only		Ao=NA Bo=NA Ho=Ai+Bi	Ao=Hi Bo=NA Ho=Ai+Bi	Ao=Hi Bo=NA Ho=Ai+Bi	Ao=NA Bo=NA Ho=Bi	Ao=Ai Bo=NA Ho=Bi
	send recv			Ao=Hi Bo=Hi Ho=Ai+Bi	Ao=Hi Bo=Hi Ho=Ai+Bi	Ao=NA Bo=Hi Ho=Bi	Ao=Ai Bo=Hi Ho=Bi
	confrnce				Ao=Hi+Bi Bo=Hi+Ai Ho=Ai+Bi	Ao=NA Bo=Hi Ho=Bi	Ao=Ai Bo=Hi Ho=Bi
	Inactive					Ao=NA Bo=NA Ho=NA	Ao=Ai Bo=NA Ho=NA
	netw						Ao=Ai Bo=Bi Ho=NA

If there are three or more "active" connections they will still interact as defined in the table above with the outgoing media streams mixed for each interaction (union of all streams). If internal resources are used up and the streams cannot be mixed, the gateway MUST return an error (error code 403 or 502, not enough resources, are RECOMMENDED).

Appendix E: Endpoint Naming Conventions

The following sections provide some RECOMMENDED endpoint naming conventions.

E.1 Analog Access Line Endpoints

The string "aaln", should be used as the first term in a local endpoint name for analog access line endpoints. Terms following "aaln" should follow the physical hierarchy of the gateway so that if the gateway has a number of RJ11 ports, the local endpoint name could look like the following:

```
aaln/#
```

where "#" is the number of the analog line (RJ11 port) on the gateway.

On the other hand, the gateway may have a number of physical plug-in units, each of which contain some number of RJ11 ports, in which case, the local endpoint name might look like the following:

```
aaln/<unit #>/#
```

where <unit #> is the number of the plug in unit in the gateway and "#" is the number of the analog line (RJ11 port) on that unit. Leading zeroes MUST NOT be used in any of the numbers ("#") above.

E.2 Digital Trunks

The string "ds" should be used for the first term of digital endpoints with a naming convention that follows the physical and digital hierarchy such as:

```
ds/<unit-type1>-<unit #>/<unit-type2>-<unit #>/.../<channel #>
```

where: <unit-type> identifies the particular hierarchy level. Some example values of <unit-type> are: "s", "su", "oc3", "ds3", "e3", "ds2", "e2", "ds1", "e1" where "s" indicates a slot number and "su" indicates a sub-unit within a slot. Leading zeroes MUST NOT be used in any of the numbers ("#") above.

The <unit #> is a decimal number which is used to reference a particular instance of a <unit-type> at that level of the hierarchy. The number of levels and naming of those levels is based on the physical hierarchy within the media gateway.

E.3 Virtual Endpoints

Another type of endpoint is one that is not associated with a physical interface (such as an analog or digital endpoint). This type of endpoint is called a virtual endpoint and is often used to represent some DSP resources that gives the endpoint some capability. Examples are announcement, IVR or conference bridge devices. These devices may have multiple instances of DSP functions so that a possible naming convention is:

<virtual-endpoint-type>/<endpoint-#>

where <virtual-endpoint-type> may be some string representing the type of endpoint (such as "ann" for announcement server or "cnf" for conference server) and <endpoint-#> would identify a particular virtual endpoint within the device. Leading zeroes MUST NOT be used in the number ("#") above. If the physical hierarchy of the server includes plug-in DSP cards, another level of hierarchy in the local endpoint name may be used to describe the plug in unit.

A virtual endpoint may be created as the result of using the "any of" wildcard. Similarly, a virtual endpoint may cease to exist once the last connection on the virtual endpoint is deleted. The definition of the virtual endpoint MUST detail both of these aspects.

When a <virtual-endpoint-type> creates and deletes virtual endpoints automatically, there will be cases where no virtual endpoints exist at the time a RestartInProgress command is to be issued. In such cases, the gateway SHOULD simply use the "all of" wildcard in lieu of any specific <endpoint-#> as in, e.g.:

ann/*@mygateway.whatever.net

If the RestartInProgress command refers to all endpoints in the gateway (virtual or not), the <virtual-endpoint-id> can be omitted as in, e.g.:

*@mygateway.whatever.net

Commands received by the gateway will still have to refer to an actual endpoint (possibly created by that command by use of the "any of" wildcard) in order for the command to be processed though.

E.4 Media Gateway

MGCP only defines operation on endpoints in a media gateway. It may be beneficial to define an endpoint that represents the gateway itself as opposed to the endpoints managed by the gateway. Implementations that wish to do so should use the local endpoint name "mg" (for media gateway) as in:

```
mg@mygateway.whatever.net
```

Note that defining such an endpoint does not change any of the protocol semantics, i.e., the "mg" endpoint and other endpoints (e.g., digital trunks) in the gateway are still independent endpoints and MUST be treated as such. For example, RestartInProgress commands MUST still be issued for all endpoints in the gateway as usual.

E.5 Range Wildcards

As described in Section 2.1.2, the MGCP endpoint naming scheme defines the "all of" and "any of" wildcards for the individual terms in a local endpoint name. While the "all of" wildcard is very useful for reducing the number of messages, it can by definition only be used when we wish to refer to all instances of a given term in the local endpoint name. Furthermore, in the case where a command is to be sent by the gateway to the Call Agent, the "all of" wildcard can only be used if all of the endpoints named by it have the same "notified entity". Implementations that prefer a finer-grained wildcarding scheme can use the range wildcarding scheme described here.

A range wildcard is defined as follows:

```
RangeWildcard    = "[" NumericalRange *( "," NumericalRange ) "]"
NumericalRange   = 1*(DIGIT) [ "-" 1*(DIGIT) ]
```

Note that white space is not permitted. Also, since range wildcards use the character "[" to indicate the start of a range, the "[" character MUST NOT be used in endpoint names that use range wildcards. The length of a range wildcard SHOULD be bounded to a reasonably small value, e.g., 128 characters.

Range wildcards can be used anywhere an "all of" wildcard can be used. The semantics are identical for the endpoints named. However, it MUST be noted, that use of the range wildcarding scheme requires support on both the gateway and the Call Agent. Therefore, a gateway MUST NOT assume that it's Call Agent supports range wildcarding and vice versa. In practice, this typically means that both the gateway and Call Agent will need to be provisioned consistently in order to

use range wildcards. Also, if a gateway or Call Agent using range wildcards receives an error response that could indicate a possible endpoint naming problem, they MUST be able to automatically revert to not using range wildcards.

The following examples illustrates the use of range wildcards:

```
ds/ds1-1/[1-12]
ds/ds1-1/[1,3,20-24]
ds/ds1-[1-2]/*
ds/ds3-1/[1-96]
```

The following example illustrates how to use it in a command:

```
RSIP 1204 ds/ds3-1/[1-96]@tgw-18.whatever.net MGCP 1.0
RM: restart
RD: 0
```

Appendix F: Example Command Encodings

This appendix provides examples of commands and responses shown with the actual encoding used. Examples are provided for each command. All commentary shown in the commands and responses is optional.

F.1 NotificationRequest

The first example illustrates a NotificationRequest that will ring a phone and look for an off-hook event:

```
RQNT 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: l/hd(N)
S: l/rg
```

The response indicates that the transaction was successful:

```
200 1201 OK
```

The second example illustrates a NotificationRequest that will look for and accumulate an off-hook event, and then provide dial-tone and accumulate digits according to the digit map provided. The "notified entity" is set to "ca@ca1.whatever.net:5678", and since the SignalRequests parameter is empty (it could have been omitted as well), all currently active TO signals will be stopped. All events in the quarantine buffer will be processed, and the list of events to detect in the "notification" state will include fax tones in addition to the "requested events" and persistent events:

```
RQNT 1202 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: L/hd(A, E(S(L/dl),R(L/oc, L/hu, D/[0-9#*T](D))))
D: (0T|00T|#xxxxxxx|*xx|91xxxxxxxxxx|9011x.T)
S:
Q: process
T: G/ft
```

The response indicates that the transaction was successful:

```
200 1202 OK
```

F.2 Notify

The example below illustrates a Notify message that notifies an off-hook event followed by a 12-digit number beginning with "91". A transaction identifier correlating the Notify with the NotificationRequest it results from is included. The command is sent to the current "notified entity", which typically will be the actual value supplied in the NotifiedEntity parameter, i.e., "ca@ca1.whatever.net:5678" - a failover situation could have changed this:

```
NTFY 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: L/hd,D/9,D/1,D/2,D/0,D/1,D/8,D/2,D/9,D/4,D/2,D/6,D/6
```

The Notify response indicates that the transaction was successful:

```
200 2002 OK
```

F.3 CreateConnection

The first example illustrates a CreateConnection command to create a connection on the endpoint specified. The connection will be part of the specified CallId. The LocalConnectionOptions specify that G.711 mu-law will be the codec used and the packetization period will be 10 ms. The connection mode will be "receive only":

```
CRCX 1204 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
```

The response indicates that the transaction was successful, and a connection identifier for the newly created connection is therefore included. A session description for the new connection is included as well - note that it is preceded by an empty line.

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The second example illustrates a CreateConnection command containing a notification request and a RemoteConnectionDescriptor:

```
CRCX 1205 aaln/1@rgw-2569.whatever.net MGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789AD
R: L/hd
S: L/rg

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction failed, because the phone was already off-hook. Consequently, neither a connection-id nor a session description is returned:

```
401 1205 Phone off-hook
```

Our third example illustrates the use of the provisional response and the three-way handshake. We create another connection and acknowledge the previous response received by using the response acknowledgement parameter:

```
CRCX 1206 aaln/1@rgw-2569.whatever.net MGCP 1.0
K: 1205
C: A3C47F21456789F0
L: p:10, a:PCMU
M: inactive

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

A provisional response is returned initially:

```
100 1206 Pending
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

A little later, the final response is received:

```
200 1206 OK
K:
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The Call Agent acknowledges the final response as requested:

```
000 1206
```

and the transaction is complete.

F.4 ModifyConnection

The first example shows a ModifyConnection command that simply sets the connection mode of a connection to "send/receive" - the "notified entity" is set as well:

```
MDCX 1209 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
N: ca@ca1.whatever.net
M: sendrecv
```

The response indicates that the transaction was successful:

```
200 1209 OK
```

In the second example, we pass a session description and include a notification request with the ModifyConnection command. The endpoint will start playing ring-back tones to the user:

```
MDCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: L/hu
S: G/rt

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction was successful:

```
200 1206 OK
```

F.5 DeleteConnection (from the Call Agent)

In this example, the Call Agent simply instructs the gateway to delete the connection "FDE234C8" on the endpoint specified:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The response indicates success, and that the connection was deleted. Connection parameters for the connection are therefore included as well:

```
250 1210 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

F.6 DeleteConnection (from the gateway)

In this example, the gateway sends a DeleteConnection command to the Call Agent to instruct it that a connection on the specified endpoint has been deleted. The ReasonCode specifies the reason for the deletion, and Connection Parameters for the connection are provided as well:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
E: 900 - Hardware error
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

The Call Agent sends a success response to the gateway:

```
200 1210 OK
```

F.7 DeleteConnection (multiple connections from the Call Agent)

In the first example, the Call Agent instructs the gateway to delete all connections related to call "A3C47F21456789F0" on the specified endpoint:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
```

The response indicates success and that the connection(s) were deleted:

```
250 1210 OK
```

In the second example, the Call Agent instructs the gateway to delete all connections related to all of the endpoints specified:

```
DLCX 1210 aaln/*@rgw-2567.whatever.net MGCP 1.0
```

The response indicates success:

```
250 1210 OK
```

F.8 AuditEndpoint

In the first example, the Call Agent wants to learn what endpoints are present on the gateway specified, hence the use of the "all of" wild-card for the local portion of the endpoint-name:

```
AUEP 1200 *@rgw-2567.whatever.net MGCP 1.0
```

The gateway indicates success and includes a list of endpoint names:

```
200 1200 OK
Z: aaln/1@rgw-2567.whatever.net
Z: aaln/2@rgw-2567.whatever.net
```

In the second example, the capabilities of one of the endpoints is requested:

```
AUEP 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0
F: A
```

The response indicates success and the capabilities as well. Two codecs are supported, however with different capabilities. Consequently two separate capability sets are returned:

```
200 1201 OK
A: a:PCMU, p:10-100, e:on, s:off, v:L;S, m:sendonly;
         recvonly;sendrecv;inactive;netwloop;netwtest
A: a:G729, p:30-90, e:on, s:on, v:L;S, m:sendonly;
         recvonly;sendrecv;inactive;confrnce;netwloop
```

Note that the carriage return in the Capabilities lines are shown for formatting reasons only - they are not permissible in a real implementation.

In the third example, the Call Agent audits several types of information for the endpoint:

```
AUEP 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0
F: R,D,S,X,N,I,T,O,ES
```

The response indicates success:

```
200 2002 OK
R: L/hu,L/oc(N),D/[0-9](N)
D:
S: L/vmwi(+)
X: 0123456789B1
N: [128.96.41.12]
I: 32F345E2
T: G/ft
O: L/hd,D/9,D/1,D/2
ES: L/hd
```

The list of requested events contains three events. Where no package name is specified, the default package is assumed. The same goes for actions, so the default action - Notify - must therefore be assumed for the "L/hu" event. The omission of a value for the "digit map" means the endpoint currently does not have a digit map. There are currently no active time-out signals, however the OO signal "vmwi" is currently on and is consequently included - in this case it was parameterized, however the parameter could have been excluded. The current "notified entity" refers to an IP-address and only a single connection exists for the endpoint. The current value of DetectEvents is "G/ft", and the list of ObservedEvents contains the four events specified. Finally, the event-states audited reveals that the phone was off-hook at the time the transaction was processed.

F.9 AuditConnection

The first example shows an AuditConnection command where we audit the CallId, NotifiedEntity, LocalConnectionOptions, Connection Mode, LocalConnectionDescriptor, and the Connection Parameters:

```
AUCX 2003 aaln/1@rgw-2567.whatever.net MGCP 1.0
I: 32F345E2
F: C,N,L,M,LC,P
```

The response indicates success and includes information for the RequestedInfo:

```
200 2003 OK
C: A3C47F21456789F0
N: ca@ca1.whatever.net
L: p:10, a:PCMU
M: sendrecv
P: PS=395, OS=22850, PR=615, OR=30937, PL=7, JI=26, LA=47

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0
```

In the second example, we request to audit RemoteConnectionDescriptor and LocalConnectionDescriptor:

```
AUCX 1203 aaln/2@rgw-2567.whatever.net MGCP 1.0
I: FDE234C8
F: RC,LC
```

The response indicates success, and includes information for the RequestedInfo. In this case, no RemoteConnectionDescriptor exists, hence only the protocol version field is included for the RemoteConnectionDescriptor:

```
200 1203 OK

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0

v=0
```

F.10 RestartInProgress

The first example illustrates a RestartInProgress message sent by an gateway to inform the Call Agent that the specified endpoint will be taken out-of-service in 300 seconds:

```
RSIP 1200 aaln/1@rgw-2567.whatever.net MGCP 1.0
RM: graceful
RD: 300
```

The Call Agent's response indicates that the transaction was successful:

```
200 1200 OK
```

In the second example, the RestartInProgress message sent by the gateway informs the Call Agent, that all of the gateway's endpoints are being placed in-service in 0 seconds, i.e., they are currently in service. The restart delay could have been omitted as well:

```
RSIP 1204 *@rgw-2567.whatever.net MGCP 1.0
RM: restart
RD: 0
```

The Call Agent's response indicates success, and furthermore provides the endpoints in question with a new "notified entity":

```
200 1204 OK
N: CA-1@whatever.net
```

Alternatively, the command could have failed with a new "notified entity" as in:

```
521 1204 OK
N: CA-1@whatever.net
```

In that case, the command would then have to be retried in order to satisfy the "restart procedure", this time going to Call Agent "CA-1@whatever.net".

Appendix G: Example Call Flows

The message flow tables in this section use the following abbreviations:

* rgw = Residential Gateway

* ca = Call Agent

* n+ = step 'n' is repeated one or more times

Note that any use of upper and lower case within the text of the messages is to aid readability and is not in any way a requirement. The only requirement involving case is to be case insensitive at all times.

G.1 Restart

G.1.1 Residential Gateway Restart

The following table shows a message sequence that might occur when a call agent (ca) is contacted by two independent residential gateways (rgw1 and rgw2) which have restarted.

Table F.1: Residential Gateway Restart

step#	usr1	rgw1	ca	rgw2	usr2
1		rsip ->			
			<- ack		
2			<- auep		
		ack ->			
3+			<- rqnt		
		ack ->			
4				<- rsip	
			ack ->		
5			auep ->		
				<- ack	
6+			rqnt ->		
				<- ack	

Step 1 - RestartInProgress (rsip) from rgw1 to ca

rgw1 uses DNS to determine the domain name of ca and send to the default port of 2727. The command consists of the following:

```
rsip 1 *@rgw1.whatever.net mgcp 1.0
rm: restart
```

The "*" is used to inform ca that all endpoints of rgw1 are being restarted, and "restart" is specified as the restart method. The Call Agent "ca" acknowledges the command with an acknowledgement message containing the transaction-id (in this case 1) for the command. It sends the acknowledgement to rgw1 using the same port specified as the source port for the rsip. If none was indicated, it uses the default port of 2727.

```
200 1 ok
```

A response code is mandatory. In this case, "200", indicates "the requested transaction was executed normally". The response string is optional. In this case, "ok" is included as an additional description.

Step 2 - AuditEndpoint (auep) from ca to rgw1

The command consists of the following:

```
auep 153 *@rgw1.whatever.net mgcp 1.0
```

The "*" is used to request audit information from rgw1 of all its endpoints. rgw1 acknowledges the command with an acknowledgement message containing the transaction-id (in this case 153) of the command, and it includes a list of its endpoints. In this example, rgw1 has two endpoints, aaln/1 and aaln/2.

```
200 153 ok
Z: aaln/1@rgw1.whatever.net
Z: aaln/2@rgw1.whatever.net
```

Once it has the list of endpoint ids, ca may send individual AuditEndpoint commands in which the "*" is replaced by the id of the given endpoint. As its response, rgw1 would replace the endpoint id list returned in the example with the info requested for the endpoint. This optional message exchange is not shown in this example.

Step 3 - NotificationRequest (rqnt) from ca to each endpoint of rgw1

In this case, ca sends two rqnts, one for aaln/1:

```
rqnt 154 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a0
```

and a second for aaln/2:

```
rqnt 155 aaln/2@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a1
```

Note that in the requested events parameter line, the event is fully specified as "l/hd", i.e., with the package name, in order to avoid any potential ambiguity. This is the recommended behavior. For the sake of clarity, the action, which in this case is to Notify, is explicitly specified by including the "(n)". If no action is specified, Notify is assumed as the default regardless of the event. If any other action is desired, it must be stated explicitly.

The expected response from rgw1 to these requests is an acknowledgement from aaln/1 as follows:

```
200 154 ok
```

and from aaln/2:

```
200 155 ok
```

Step 4 RestartInProgress (rsip) from rgw2 to ca

```
rsip 0 *@rgw2.whatever.net mgcp 1.0
rm: restart
```

followed by the acknowledgement from ca:

```
200 0 ok
```

Step 5 - AuditEndpoint (auep) from ca to rgw2

```
auep 156 *@rgw2.whatever.net mgcp 1.0
```

followed by an acknowledgement from rgw2:

```
200 156 ok
z: aaln/1@rgw2.whatever.net
z: aaln/2@rgw2.whatever.net
```

Step 6 - NotificationRequest (rqnt) from ca to each endpoint of rgw2

```
rqnt 157 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a2
```

followed by:

```
rqnt 158 aaln/2@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a3
```

with rgw2 acknowledging for aaln/1:

```
200 157 ok
```

and for aaln/2:

```
200 158 ok
```

G.1.2 Call Agent Restart

The following table shows the message sequence which occurs when a call agent (ca) restarts. How it determines the address information of the gateways, in this case rgw1 and rgw2, is not covered in this document. For interoperability, it is RECOMMENDED to provide the ability to configure the call agent to send AUEP (*) to specific addresses and ports.

Table F.2: Residential Gateway Restart

#	usr1	rgw1	ca	rgw2	usr2
1			<- auep		
		ack ->			
2+			<- rqnt		
		ack ->			
3			auep ->		
				<- ack	
4+			rqnt ->		
				<- ack	

Step 1 - AuditEndpoint (auep) from ca to rgw1

The command consists of the following:

```
auep 0 *@rgw1.whatever.net mgcp 1.0
```

The "*" is used to request audit information from rgw1 of all its endpoints. rgw1 acknowledges the command with an acknowledgement message containing the transaction id (in this case 0) of the command, and it includes a list of its endpoints. In this example, rgw1 has two endpoints, aaln/1 and aaln/2.

```
200 0 ok
z: aaln/1@rgw1.whatever.net
z: aaln/2@rgw1.whatever.net
```

Once it has the list of endpoint ids, ca may send individual AuditEndpoint commands in which the "*" is replaced by the id of the given endpoint. As its response, rgw1 would replace the endpoint id list returned in the example with the info requested for the endpoint. This optional message exchange is not shown in this example.

Step 2 - NotificationRequest (rqnt) off-hook from ca to rgw1

In this case, ca sends two rqnts, one for aaln/1:

```
rqnt 1 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567890
```

and a second for aaln/2:

```
rqnt 2 aaln/2@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567891
```

The expected response from rgw1 to these requests is an acknowledgement from aaln/1 as follows:

```
200 1 ok
```

and from aaln/2:

```
200 2 ok
```

Step 3 - AuditEndpoint (auep) from ca to rgw2

```
auep 3 *@rgw2.whatever.net mgcp 1.0
```

followed by an acknowledgement from rgw2:

```
200 3 ok
z: aaln/1@rgw2.whatever.net
z: aaln/2@rgw2.whatever.net
```

Step 4 - NotificationRequest (rqnt) from ca to each endpoint of rgw2

```
rqnt 4 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567892
```

followed by:

```
rqnt 5 aaln/2@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567893
```

with rgw2 acknowledging for aaln/1:

```
200 4 ok
```

and for aaln/2:

```
200 5 ok
```

G.2 Connection Creation

G.2.1 Residential Gateway to Residential Gateway

The following table shows the message sequence which occurs when a user (usr1) makes a call through a residential gateway (rgw1) to a user served by another residential gateway (rgw2). This example illustrates the communication between the residential gateways and the call agent (ca) only. The local name of the endpoints in this example is aaln/1 for both gateways, and references within the description of the steps to rgw1 and rgw2 can be assumed to refer to aaln/1 of rgw1 and aaln/1 of rgw2. Note that this is only an example and is not the only legal call scenario.

Table F.3: Residential Gateway Connection Creation

#	usr1	rgw1	ca	rgw2	usr2
1	offhook ->	ntfy ->			
			<- ack		
2	<- dialtone		<- rqnt		
		ack ->			
3	digits ->	ntfy ->			
			<- ack		
4			<- rqnt		
		ack ->			
5	<- recvonly		<- crcx		
		ack ->			
6			crcx ->		sendrcv ->
				<- ack	
7	<- recvonly		<- mdcx		
		ack ->			
8	<- ringback		<- rqnt		
		ack ->			
9			rqnt ->		ringing ->
				<- ack	
10				<- ntfy	<- offhook
			ack ->		
11			rqnt ->		
				<- ack	
12			<- rqnt		
		ack ->			
13	<- sendrcv		<- mdcx		
		ack ->			

Step 1 - Notify (ntfy) offhook from rgw1 to ca

This ntfy is the result of usr1 going offhook and assumes ca had previously sent an rqnt with RequestId "445678944" to rgw1 requesting notification in the event of an offhook:

```
ntfy 12 aaln/1@rgw1.whatever.net mgcp 1.0
o: l/hd
x: 445678944
```

Acknowledgement from ca:

```
200 12 ok
```

Step 2 - Request Notification (rqnt) for digits from ca to rgw1

Request rgw1 to notify if on-hook and collect digits according to the digit map, and to provide dialtone:

```
rqnt 1057 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n), d/[0-9#*T](d)
s: l/dl
x: 445678945
d: 5xxx
```

Acknowledgement from rgw1:

```
200 1057 ok
```

Step 3 - Notify (ntfy) digits from rgw1 to ca

```
ntfy 13 aaln/1@rgw1.whatever.net mgcp 1.0
o: d/5, d/0, d/0, d/1
x: 445678945
```

Acknowledgement from ca:

```
200 13 ok
```

Step 4 - Request Notification (rqnt) from ca to rgw1

Request rgw1 to notify in the event of an on-hook transition:

```
rqnt 1058 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678946
```

Acknowledgement from rgw1:

```
200 1058 ok
```

Step 5 - Create Connection (crcx) from ca to rgw1

Request a new connection on rgw1 with the specified local connection options, including 20 msec as the packetization period, G.711 mu-law as the codec, and receive only as the mode:

```
crcx 1059 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
l: p:20, a:PCMU
m: recvonly
```

Acknowledgement from rgw1 that a new connection, "456789fedcba5", has been created, followed by a blank line and then the SDP parameters:

```
200 1059 ok
i: 456789fedcba5

v=0
o=- 23456789 98765432 IN IP4 192.168.5.7
s=-
c=IN IP4 192.168.5.7
t=0 0
m=audio 6058 RTP/AVP 0
```

Step 6 - Create Connection (crcx) from ca to rgw2

Request a new connection on rgw2. The request includes the session description returned by rgw1 such that a two way connection can be initiated:

```
crcx 2052 aaln/1@rgw2.whatever.net mgcp 1.0
c: 9876543210abcdef
l: p:20, a:PCMU
m: sendrecv

v=0
o=- 23456789 98765432 IN IP4 192.168.5.7
s=-
c=IN IP4 192.168.5.7
t=0 0
m=audio 6058 RTP/AVP 0
```

Acknowledgement from rgw2 that a new connection, "67890af54c9", has been created; followed by a blank line and then the SDP parameters:

```
200 2052 ok
i: 67890af54c9

v=0
o=- 23456889 98865432 IN IP4 192.168.5.8
s=-
c=IN IP4 192.168.5.8
t=0 0
m=audio 6166 RTP/AVP 0
```

Step 7 - Modify Connection (mdcx) from ca to rgw1

Request rgw1 to modify the existing connection, "456789fedcba5", to use the session description returned by rgw2 establishing a half duplex connection which, though not used in this example, could be used to provide usr1 with in band ringback tone, announcements, etc:

```
mdcx 1060 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
l: p:20, a:PCMU
M: recvonly

v=0
o=- 23456889 98865432 IN IP4 192.168.5.8
s=-
c=IN IP4 192.168.5.8
t=0 0
m=audio 6166 RTP/AVP 0
```

Acknowledgement from rgw1:

```
200 1060 ok
```

Step 8 - Request Notification (rqnt) from ca for rgw1 to provide ringback

Request rgw1 to notify in the event of an on-hook transition, and also to provide ringback tone:

```
rqnt 1061 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
s: g/rt
x: 445678947
```

Acknowledgement from rgw1:

```
200 1061 ok
```

Step 9 - Request Notification (rqnt) from ca to rgw2 to provide ringing

Request rgw2 to continue to look for offhook and provide ringing:

```
rqnt 2053 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
s: l/rg
x: 445678948
```

Acknowledgement from rgw2:

```
200 2053 ok
```

Step 10 - Notify (ntfy) offhook from rgw2 to ca

```
ntfy 27 aaln/1@rgw2.whatever.net mgcp 1.0
o: l/hd
x: 445678948
```

Acknowledgement from ca:

```
200 27 ok
```

Step 11 - Request Notification (rqnt) of on-hook from ca to rgw2

```
rqnt 2054 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678949
```

Acknowledgement from rgw2:

```
200 2054 ok
```

Step 12 - Request Notification (rqnt) of on-hook from ca to rgw1

```
rqnt 1062 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678950
```

Acknowledgement from rgw1:

```
200 1062 ok
```

Step 13 - Modify Connection (mdcx) from ca to rgw1

Request rgw1 to modify the existing connection, "456789fedcba5", to sendrecv such that a full duplex connection is initiated:

```
mdcx 1063 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
m: sendrecv
```

Acknowledgement from rgw1:

```
200 1063 ok
```

G.3 Connection Deletion

G.3.1 Residential Gateway to Residential Gateway

The following table shows the message sequence which occurs when a user (usr2) initiates the deletion of an existing connection on a residential gateway (rgw2) with a user served by another residential gateway (rgw1). This example illustrates the communication between the residential gateways and the call agent (ca) only. The local name of the endpoints in this example is aaln/1 for both gateways, and references within the description of the steps to rgw1 and rgw2 can be assumed to refer to aaln/1 of rgw1 and aaln/1 of rgw2.

Table F.4: Residential Gateway Connection Deletion

#	usr1	rgw1	ca	rgw2	usr2
1			ack ->	<- ntfy	<- on-hook
2			dlcx ->	<- ack	
3		ack ->	<- dlcx		
4			rqnt ->	<- ack	
5	on-hook ->	ntfy ->	<- ack		
6		ack ->	<- rqnt		

Step 1 - Notify (ntfy) offhook from rgw1 to ca

This ntfy is the result of usr2 going on-hook and assumes that ca had previously sent an rqnt to rgw2 requesting notification in the event of an on-hook (see end of Connection Creation sequence):

```
ntfy 28 aaln/1@rgw2.whatever.net mgcp 1.0
o: l/hu
x: 445678949
```

Acknowledgement from ca:

```
200 28 ok
```

Step 2 - Delete Connection (dlcx) from ca to rgw2

Requests rgw2 to delete the connection "67890af54c9":

```
dlcx 2055 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 67890af54c9
```

Acknowledgement from rgw2. Note the response code of "250" meaning "the connection was deleted":

```
250 2055 ok
```

Step 3 - Delete Connection (dlcx) from ca to rgw1

Requests rgw1 to delete the connection "456789fedcba5":

```
dlcx 1064 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
```

Acknowledgement from rgw1:

```
250 1064 ok
```

Step 4 - NotificationRequest (rqnt) from ca to rgw2

Requests rgw2 to notify ca in the event of an offhook transition:

```
rqnt 2056 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 445678951
```

Acknowledgement from rgw2:

```
200 2056 ok
```

Step 5 - Notify (ntfy) on-hook from rgw1 to ca

Notify ca that usr1 at rgw1 went back on-hook:

```
ntfy 15 aaln/1@rgw1.whatever.net mgcp 1.0
o: l/hu
x: 445678950
```

Acknowledgement from ca:

```
200 15 ok
```

Step 6 - NotificationRequest (rqnt) offhook from ca to rgw1

Requests rgw1 to notify ca in the event of an offhook transition:

```
rqnt 1065 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 445678952
```

Acknowledgement from rgw1:

200 1065 ok

Authors' Addresses

Flemming Andreasen
Cisco Systems
499 Thornall Street, 8th Floor
Edison, NJ 08837

EMail: fandreas@cisco.com

Bill Foster
Cisco Systems
771 Alder Drive
Milpitas, CA 95035

EMail: bfoster@cisco.com

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.