

PacketCable™ Superseded

PacketCable™ Network-Based Call Signaling Protocol Specification

Pkt-SP-EC-MGCP-I01-990312

March 12, 1999

**INTERIM
SPECIFICATION**

Notice

This document is a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the PacketCable™ initiative, and the cable industry in general. Neither CableLabs, nor any other entity participating in the creation of this document is responsible for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document by any party. This document is furnished on an AS-IS basis and neither CableLabs, nor other participating entity provides any representation or warranty, express or implied, regarding its accuracy, completeness, or fitness for a particular purpose. Distribution of this document is restricted pursuant to the terms of separate access agreements negotiated with each of the participating entities.

© Copyright 1999 Cable Television Laboratories, Inc.

All rights reserved.

Abstract

This specification describes a profile of the Media Gateway Control Protocol (MGCP) for PacketCable™ embedded clients, which we will refer to as the PacketCable™ Network-based Call Signaling (NCS) protocol. MGCP is a call signaling protocol for use in a centralized call control architecture, and assumes relatively simple client devices. The call signaling protocol is one layer of the overall PacketCable™ suite of specifications and relies upon companion protocol specifications to provide complete end-to-end PacketCable™ functionality. The scope of NCS is currently only embedded Voice over IP client devices in a PacketCable environment and the NCS profile has therefore simplified and in some cases modified the base MGCP 0.1 draft accordingly. Support for video will be added in a later version of this document.

Document Status Sheet

Document Control Number:	Pkt-SP-EC-MGCP-I01-990312			
Document Title:	PacketCable™ Network-Based Call Signaling Protocol Specification			
Revision History:	<p>D1 Draft 0.8 of the SGCP Variant specification – upgraded to Draft status for distribution to NDA vendors.</p> <p>D2 Draft renamed and now based on MGCP</p> <p>I1 Interim Release 1.0, based on 11 review/revision cycles</p>			
Date:	December 10, 2003			
Reference:	PacketCable™ Network-Based Call Signaling Protocol Specification			
Responsible Author:	PacketCable Embedded Client Signaling Focus Team			
Status:	Work in Progress	Draft	Interim	Released
Distribution Restrictions:	Author Only	CL/Member	CL/ PacketCable/ Vendor	Public

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback, that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Interim	A document which has undergone rigorous Member and vendor review, suitable for use by vendors to design in conformance to and for field testing.
Released	A stable document, reviewed, tested and validated, suitable to enable cross-vendor interoperability.

Contents

1	STATUS OF THIS DOCUMENT	1
2	ABSTRACT	2
2.1	Relation With H.323 Standards	5
2.2	Relation With IETF Standards	5
3	MEDIA GATEWAY CONTROL INTERFACE (MGCI)	7
3.1	Model and Naming Conventions	7
3.1.1	<i>Endpoint Names</i>	7
3.1.1.1	Embedded Client Endpoint Names	8
3.1.1.1.1	Analog Access Line Endpoints	8
3.1.1.1.2	Video Endpoints	9
3.1.2	<i>Call Names</i>	10
3.1.3	<i>Connection Names</i>	11
3.1.4	<i>Names of Call Agents and Other Entities</i>	11
3.1.5	<i>Digit Maps</i>	12
3.1.6	<i>Events and Signals</i>	14
3.2	SDP Use	16
3.3	Gateway Control Functions	16
3.3.1	<i>NotificationRequest</i>	18
3.3.2	<i>Notifications</i>	24
3.3.3	<i>CreateConnection</i>	26
3.3.4	<i>ModifyConnection</i>	30
3.3.5	<i>DeleteConnection (From the Call Agent)</i>	31
3.3.6	<i>DeleteConnection (From the Embedded Client)</i>	33
3.3.7	<i>DeleteConnection (Multiple Connections From the Call Agent)</i>	34
3.3.8	<i>Auditing</i>	35
3.3.8.1	<i>AuditEndPoint</i>	35
3.3.8.2	<i>AuditConnection</i>	38
3.3.9	<i>Restart in Progress</i>	39
3.4	States, Failover and Race Conditions	40
3.4.1	<i>Recaps and Highlights</i>	41
3.4.2	<i>Retransmission, and Detection of Lost Associations</i>	42

3.4.3	<i>Race Conditions</i>	45
3.4.3.1	Quarantine list.....	45
3.4.3.2	Explicit detection	48
3.4.3.3	Ordering of commands, and treatment of disorder	49
3.4.3.4	Fighting the restart avalanche.....	50
3.4.3.5	Disconnected Endpoints	51
3.5	Return Codes and Error Codes	52
3.6	Reason Codes	54
4	MEDIA GATEWAY CONTROL PROTOCOL	55
4.1	General Description.....	55
4.2	Command Header.....	56
4.2.1	<i>Command Line</i>	56
4.2.1.1	Requested Verb Coding	56
4.2.1.2	Transaction Identifiers.....	57
4.2.1.3	Endpoint, Call Agent and NotifiedEntity Name Coding	57
4.2.1.4	Protocol Version Coding	58
4.2.2	<i>Parameter Lines</i>	58
4.2.2.1	RequestIdentifier	61
4.2.2.2	Local Connection Options.....	61
4.2.2.3	Capabilities.....	61
4.2.2.4	Connection Parameters.....	62
4.2.2.5	Reason Codes	63
4.2.2.6	Connection Mode	63
4.2.2.7	Event/Signal Name Coding	64
4.2.2.8	RequestedEvents	65
4.2.2.9	SignalRequests	66
4.2.2.10	ObservedEvents.....	66
4.2.2.11	RequestedInfo	67
4.2.2.12	QuarantineHandling	67
4.2.2.13	DetectEvents	67
4.2.2.14	EventStates.....	67
4.2.2.15	RestartMethod.....	68
4.3	Response Header Formats	68
4.3.1	<i>CreateConnection</i>	69
4.3.2	<i>ModifyConnection</i>	69
4.3.3	<i>DeleteConnection</i>	69

4.3.4	<i>NotificationRequest</i>	70
4.3.5	<i>Notify</i>	70
4.3.6	<i>AuditEndpoint</i>	70
4.3.7	<i>AuditConnection</i>	70
4.3.8	<i>RestartInProgress</i>	71
4.4	Session Description Encoding.....	71
4.4.1	<i>SDP Audio Service Use</i>	71
4.4.2	<i>SDP Video Service Use</i>	74
4.5	Transmission Over UDP.....	74
4.5.1	<i>Reliable Message Delivery</i>	74
4.5.2	<i>Retransmission Strategy</i>	75
4.6	Piggy-Backing.....	76
5	SECURITY	77
5.1	Using IPSEC with MGCP.....	77
6	ACKNOWLEDGEMENTS	78
7	REFERENCES	79
APPENDIX A - EVENT PACKAGES		81
A.1	Analog Access Lines.....	81
A.1.1	<i>Line Package</i>	81
A.1.2	<i>ADSI Package</i>	88
A.2	Video.....	89
APPENDIX B - MODE INTERACTIONS		90
APPENDIX C - EXAMPLE COMMAND ENCODINGS		94
C.1	NotificationRequest.....	94
C.2	Notify.....	95
C.3	CreateConnection.....	95
C.4	ModifyConnection.....	96
C.5	DeleteConnection (From the Call Agent).....	97
C.6	DeleteConnection (From the Embedded Client).....	97
C.7	DeleteConnection (Multiple Connections From the Call Agent).....	97
C.8	AuditEndpoint.....	98

C.9 – AuditConnection99

C.10 – RestartInProgress100

APPENDIX D – EXAMPLE CALL FLOW.....103

APPENDIX E - KNOWN ISSUES111

1 Status of This Document

This document is considered part of the PacketCable™ specification. The document is based on MGCP 0.1 [1], which has been submitted as an Internet-draft. Internet-drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. However, this document will not be submitted for formal review by the IETF until it is approved by the PacketCable™ project members.

2 Abstract

This document describes a PacketCable™ profile of an application programming interface (MGCI), and a corresponding protocol (MGCP) for controlling voice-over-IP (VoIP) embedded clients from external control elements. MGCP is a call control architecture that provides control intelligence to the devices and is handled by external control elements. The PacketCable™ profile, as described in this document, will be referred to as the Network-based Call Signaling (NCS) Protocol.

This document is based on the Media Gateway Control Protocol (MGCP) 0.1 Internet-Draft [1], which was the result of a merge of the Simple Gateway Control Protocol [2], and the IP Device Control (IPDC) family of protocols [3], as well as input generated by the PacketCable™ embedded client signaling team that developed this profile.

This document, which defines the PacketCable™ NCS Protocol specification, constitutes a document that is independent of MGCP in order to provide a stable reference document, while meeting current time-to-market demands for such a reference. It is the intent of this document, to be as closely aligned with MGCP as possible for the PacketCable™ environment, in order to avoid developing multiple protocols to solve the same problem. This goal has been, and continues to be, pursued through cooperation with the authors of the MGCP specification. The NCS profile of MGCP, however, is strictly and is solely defined by the contents of this document.

This NCS profile of MGCP, which will be referred to as the Network-based Call Signaling Protocol 1.0, NCS 1.0, the NCS profile, or simply NCS in this document, has been modified from the MGCP 0.1 draft in the following ways:

- *The NCS protocol only aims at supporting PacketCable™-embedded clients:* The NCS protocol supports embedded clients as defined by PacketCable™. Functionality present in the MGCP 0.1 protocol, which was superfluous to NCS, has been removed.
- *The NCS protocol contains extensions and modifications to MGCP:* PacketCable™-specific requirements have been addressed in NCS, which has resulted in a couple of minor extensions and modifications to MGCP. However, the MGCP architecture, and all of the MGCP constructs relevant to embedded clients, are preserved in NCS.
- *The NCS protocol contains minor simplifications from MGCP 0.1:* Where several choices were available, and not necessarily needed for an embedded client in the PacketCable™ environment, some simplifications have been made for embedded-client implementations.

Although MGCP is not NCS, and NCS is not MGCP, the names MGCP and NCS will be used interchangeably in this document since this document is based on MGCP. Unless otherwise stated or inferred by context, the word MGCP shall be taken to mean the NCS profile of MGCP in this document.

The document is structured in the following main sections:

- The introduction presents the basic assumptions and the relation to other protocols such as H.323, RTSP, SAP, and SIP.
- The interface section presents a conceptual overview of the NCS profile, presenting the naming conventions, the usage of the session description protocol (SDP), and the procedures that compose NCS:
 - Notification Request,
 - Notify,
 - Create Connection,
 - Modify Connection,
 - Delete Connection,
 - Audit Endpoint,
 - Audit Connection, and
 - Restart In Progress
- Each of the commands is presented as an example API with the name of the command, the parameters it can take and return, as well as the semantics of each of these. The actual encoding of the command and its parameters is shown in the protocol description section. The section concludes with a description of how reliability and race conditions are handled.
- The protocol description section presents the actual NCS encodings of the commands and parameters, which are based on simple text formats. The transmission procedure over UDP is specified as well.
- The security section presents the security of NCS, and illustrates how IP security services (IPSEC) could be used. The actual security requirements are provided in the PacketCable™ Security Specification document, which may or may not specify IPSEC as the mechanism of choice.
- The appendices contain definitions of event packages, mode interactions, example command encodings, one complete example call flow, and a list of known issues.

1 Introduction

This document describes the NCS profile of an application programming interface (MGCI) and a corresponding protocol (MGCP) for controlling embedded clients from external call control elements. An embedded client is a network element that provides:

- Two or more traditional analog (RJ11) access lines to a voice-over-IP (VoIP) network.
- Optionally, one or more video lines to a VoIP network.

Embedded clients may not be confined to residential use only. For example, they may be used in a business as well. Embedded clients are used for line-side access and, as such, are expected to have line-side equipment, e.g., analog access lines for conventional telephones associated with them, as opposed to trunk gateways.

The MGCP assumes a call control architecture where the call control “intelligence” is outside the gateways and handled by external call-control elements referred to as Call Agents. The MGCP assumes that these call-control elements, or Call Agents (CAs), will synchronize with each other to send coherent commands to the gateways under their control. The MGCP defined in this document does not define a mechanism for synchronizing Call Agents, although future PacketCable specifications may specify such mechanisms.

The MGCP assumes a connection model where the basic constructs are endpoints and connections. A gateway contains a collection of endpoints, which are sources, or sinks, of data and could be physical or virtual.

An example of a physical endpoint is 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* or an *embedded client*. Embedded clients may optionally support video as well.

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 accomplished by software. However, the NCS profile of MGCP only addresses physical endpoints.

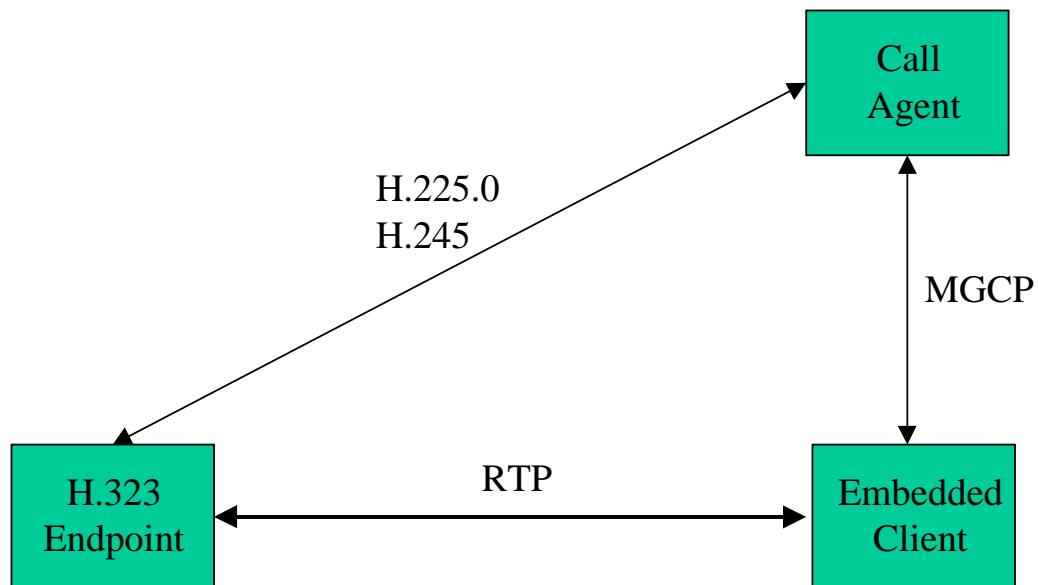
Connections are point-to-point. A point-to-point connection is an association between two endpoints with the purpose of transmitting data between these endpoints. Once this association is established for both endpoints, data transfer between these endpoints can take place.

Call Agents instruct the gateways to create connections between endpoints and to detect certain events, e.g., off-hook, and generate certain signals, e.g., ringing. It is strictly up to the Call Agent to specify how and when connections are made, between which endpoints they are made, as well as what events and signals are to be detected and generated on the endpoints. The gateway, thereby, becomes a simple device, without any call state, that

receives general instructions from the Call Agent without any need to worry about or even understand the concept of calls, call states, features, or feature interactions. When new services are introduced, customer profiles changed, etc., the changes are transparent to the gateway. The Call Agents implement the changes and generate the appropriate new mix of instructions to the gateways for the changes made. Whenever the gateway reboots, it will come up in a clean state and simply carry out the Call Agent's instructions as they are received.

2.1 Relation With H.323 Standards

The 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, which may or may not be distributed over several computer platforms, and a set of gateways. In an H.323 configuration, this distributed gateway system may interface on one side with one or more POTS lines, and on the other side with H.323 conformant systems, as illustrated below:



In the MGCP model, the gateways focus on the audio signal translation function, while the Call Agent handles the 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.

2.2 Relation With IETF Standards

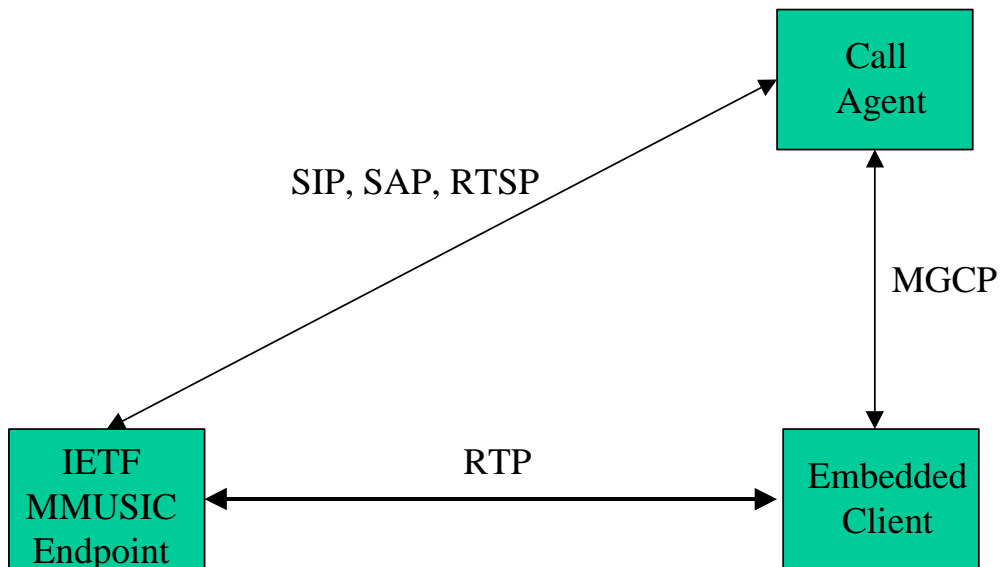
While H.323 used to be the recognized standard for VoIP terminals, the IETF also has produced specifications for other types of multi-media applications. These other specifications include:

- the session description protocol (SDP), RFC 2327,

- the session announcement protocol (SAP),
- the session initiation protocol (SIP),
- 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 and RTCP).

The distributed gateway systems and MGCP will enable PSTN telephony and embedded client users to access sessions set up using SAP, SIP, or RTSP. The Call Agent provides for signaling conversion, as illustrated below:



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.

3 Media Gateway Control Interface (MGCI)

MGCI functions provide for connection control, endpoint control, auditing, and status. They each use the same system model and the same naming conventions.

3.1 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 several Call Agents.

3.1.1 Endpoint Names

Endpoint names, a.k.a., endpoint identifiers, have two components, both of which are defined to be case insensitive here:

- the domain name of the gateway managing the endpoint,
- a local endpoint name within that gateway

Endpoint names will be 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

```
MyEmbeddedClient.cablelabs.com.
```

Also, domain-name may be an IPv4 or IPv6 address in dotted decimal form represented as a text-string and surrounded by a left and a right square bracket (“[” and “]”) as in “[128.96.41.1]” —please consult RFC 821 for details. However, use of IP addresses is generally discouraged.

Embedded clients may have one or more endpoints (e.g., one for each RJ11 jack for black phones) associated with them, and each of the endpoints is identified by a separate local endpoint name. Just like the domain-name, the local endpoint name is case insensitive. Associated with the local endpoint name is an endpoint-type, which defines the type of the endpoint, such as analog phone or video phone. The endpoint-type can be derived from the local endpoint name. The local endpoint name is a hierarchical name, where the least specific component of the name is the leftmost term, and the most specific component is the rightmost term. More formally, the local endpoint name must adhere to the following naming rules:

- The individual terms of the local endpoint name must be separated by a single slash (“/,” ASCII 2F hex).

- The individual terms are ASCII character strings composed of letters, digits or other printable characters, with the exception of characters used as delimiters in endpoint-names (“/,” “@”), characters used for wildcarding (“*,” “\$”), and white space characters.
- 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 looks like
`term1/term2/term3,`
and one of the terms of the local endpoint name is wild-carded, then the local endpoint name looks like this:
 - `*/term2/term3` if term1 is wild-carded
 - `term1/*/term3` if term2 is wild-carded
 - `term1/term2/*` if term3 is wild-carded.
In each of the examples, a dollar sign could have appeared instead of the asterisk.
- A term represented by an asterisk is to be interpreted as:
“use *all* values of this term known within the scope of the embedded client in question.”
- 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 embedded client in question.”
- Each endpoint-type may specify additional detail in the naming rules for that endpoint-type, however such rules must not be in conflict with the above.

It should be noted that different endpoint-types or even different sub-terms, e.g., “lines”, within the same endpoint-type will result in two different local endpoint names. Consequently, each “line” will be treated as a separate endpoint.

3.1.1.1 Embedded Client Endpoint Names

Endpoints in embedded clients will use the additional naming conventions specified in this section.

Embedded clients will support the following two endpoint-types:

- `aaln`. An analog access line—basically the equivalent of an analog telephone line as known in the PSTN (see [20]).
- `Video`. The details of the video device-type will be provided in a future version of this document.

3.1.1.1.1 Analog Access Line Endpoints

In addition to the naming conventions specified above, local endpoint names for endpoints of type “analog access line” (aaln) for embedded clients will adhere to the following:

- Local endpoint names contain at least one and, at most, two terms.
- Term1 will be the term “aaln” or a wildcard character. It should be noted that the use of a wildcard character for term1 can refer to any or all endpoint-types in the embedded client regardless of their type. Use of this feature is generally expected to be for administrative purposes, e.g., auditing or restart.
- Term2 will be a number from one to the number of analog access lines supported by the embedded client in question. The number thus identifies a specific analog access line on the embedded client.
- If a local endpoint name is composed of only one term, that term will be term1.
 - If term1 *is not* a wildcard character, the wildcard character dollar sign (referring to “any one”) is then assumed for term2, i.e., “aaln” is equivalent to “aaln/\$”.
 - If term1 *is* a wildcard character, the wildcard character asterisk (referring to “all”) is then assumed for term2, i.e., “*” and “\$” is equivalent to respectively “*/*” and “\$/*.”

Example analog access line local endpoint names could thus be:

- aaln/1 The first analog access line on the embedded client in question.
- aaln/2 The second analog access line on the embedded client in question.
- aaln/\$ Any analog access line on the embedded client in question.
- aaln/* All analog access lines on the embedded client in question.
- */2 The second “x line” on all endpoint-types on the embedded client in question.
- * All endpoints (regardless of endpoint-type) on the embedded client in question.

The provisioning/(auto)configuration process is responsible for obtaining and providing information about how many endpoints an embedded client has, as well as the endpoint-type of each endpoint. Although they are logically different, it should be noted that the *endpoint-type* can be derived from the local portion of the endpoint name.

3.1.1.1.2 Video Endpoints

In addition to the naming conventions specified above, local endpoint names for endpoints of type “Video” (`video`) for embedded clients will adhere to the following:

- Local endpoint names contain at least one and, at most, two terms.
- Term1 will be the term “`video`” or a wildcard character. It should be noted that the use of a wildcard character will refer to all endpoint-types in the embedded client, regardless of their type. Use of this feature is generally expected to be for administrative purposes only.
- Term2 will be a number from one to the number of “video lines” supported by the embedded client in question. The number thus identifies specific “video lines” on the embedded client.
- If a local endpoint name is composed of only one term, that term will be term1.
 - If term1 *is not* a wildcard character, the wildcard character dollar sign (referring to “any one”) is then assumed for term2.
 - If term1 *is* a wildcard character, the wildcard character asterisk (referring to “all”) is then assumed for term2.

Example video line local endpoint names could thus be:

`video/1` The first video line on the embedded client in question.

`video/2` The second video line on the embedded client in question.

`video/$` Any video line on the embedded client in question.

`video/*` All video lines on the embedded client in question.

`*/2` The second “*x* line” on all endpoint-types on the embedded client in question.

`*` All endpoints (regardless of endpoint-type) on the embedded client in question.

The provisioning/(auto)configuration process is responsible for obtaining and providing information about how many endpoints an embedded client has, as well as the endpoint-type of each endpoint. Although they are logically different, it should be noted that the *endpoint-type* can be derived from the name of the endpoint.

3.1.2 Call Names

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.

At a minimum, call identifiers **MUST** be unique within the collection of call agents that control the same gateways. However, the coordination of these call identifiers between Call Agents is outside the scope of this document. When a Call Agent builds several connections that pertain to the same call, either on the same gateway or in different gateways, these connections all will be linked to the same call through the call identifier. This identifier then can be used by accounting or management procedures, which are outside the scope of MGCP.

3.1.3 Connection Names

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** ensure that a proper waiting period, at least three minutes, elapses between the end of a connection that used this identifier and its use in a new connection for the same endpoint. The maximum length of a connection name is 32 characters.

3.1.4 Names of Call Agents and Other Entities

The Media Gateway Control Protocol has been designed for enhanced network reliability to allow implementation of redundant Call Agents. 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. The local portion of the name does not exhibit any internal structure. An example Call Agent name is:

cal@ca.whatever.net

Reliability is provided by the following precautions:

- Entities such as embedded clients 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 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 are kept in the Domain Name Service (DNS). Call Agents and gateways **MUST** keep track of the record's time-to-live 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, and when the endpoint needs to send a command to the Call Agent, it **MUST** send the command to the current “notified entity” for which endpoint(s) the command pertains. 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 reboots. If the “notified entity” for an endpoint is empty or has not been set explicitly¹, the “notified entity” will then default to the source address of the last connection handling command or notification request received for the endpoint. Auditing will thus not change the “notified entity.”

Section 3.4, p. 40 contains a more detailed description of reliability and fail-over.

3.1.5 Digit Maps

The Call Agent can ask the gateway to collect digits dialed by the user. This facility is intended to be used for analog access lines with residential gateways to collect the numbers that a user dials; it may also be used to collect access codes, credit card numbers, and other numbers requested by call control services.

An alternative procedure involves the gateway notifying the Call Agent of the dialed digits as soon as they are dialed, a.k.a., overlap sending. However, such a procedure generates a large number of interactions. It is preferable to accumulate the dialed numbers in a buffer, and then to transmit them in a single message.

The problem with this accumulation approach, however, is that it is difficult 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 load the gateway with a digit map that corresponds 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 following BNF notation:

¹ This could happen as a result of specifying an empty NotifiedEntity parameter.

Digit	::= “0” “1” “2” “3” “4” “5” “6” “7” “8” “9”	
Timer	::= “T” “t”	-- matches the detection of a timer
Letter	::= Digit Timer “#” “*” “A” “a” “B” “b” “C” “c” “D” “d”	
Range	::= “X” “x”	-- matches any digit
	“[“ Letters “]”	-- matches any of the specified letters
Letters	::= Subrange Subrange Letters	
Subrange	::= Letter	-- matches the specified letter
	Digit “-” Digit	-- matches any digit between first and last
Position	::= Letter Range	
StringElement	::= Position	-- matches an occurrence of the position
	Position “.”	-- matches an arbitrary number of occurrences
		-- of the position, including 0
String	::= StringElement StringElement String	
StringList	::= String String “ ” StringList	
DigitMap	::= String (“ StringList “)	

A DigitMap, according to this syntax, is defined either by a (case insensitive) “string” or by a “list of strings.” Regardless of the above syntax, a timer is currently only allowed if it appears in the last position in a string². Each string in the list is an alternate numbering scheme. A gateway that detects digits, letters, or timers will:

1. Add the event parameter code for the digit, letter, or timer, as a token to the end of the “current dial string” internal state variable.
2. Apply the “current dial string” to the digit map table, attempting a match, in lexical order, to each regular expression in the Digit Map.
3. If the result is under-qualified (partially matches at least one entry in the digit map), do nothing further.

² For instance, “123T” and “123[1-2T5]” satisfy that rule, but “12T3” does not.

If the result matches, or is over-qualified (i.e., no further digits could possibly produce a match), send the list of digits to the Call Agent³ and clear the “current dial string.”

Timer T is a digit input timer that can be used in two ways:

- When timer T is used with a digit map⁴, the timer is not started until the first digit is entered, and the timer is restarted after each new digit is entered until either a digit map match or mismatch occurs. In this case, timer T functions as an inter-digit timer.
- When timer T is used without a digit map, the timer is started immediately and simply cancelled (but not restarted) as soon as a digit is entered. In this case, timer T can be used as an inter-digit timer when overlap sending is used.

When used with a digit map, timer T takes on one of two values, T_{par} or T_{crit} . When at least one more digit is required for the digit string to match any of the patterns in the digit map, timer T takes on the value T_{par} , corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value T_{crit} corresponding to critical timing. When timer T is used without a digit map, timer T takes on the value T_{crit} . The default value for T_{par} is 16 seconds and the default value for T_{crit} is 4 seconds. The provisioning process may alter both of these.

Appendix A.1.1 contains additional detail and an example on use of timer T.

Digit maps can be provided to the gateway by the Call Agent, whenever the Call Agent instructs the gateway to listen for digits.

3.1.6 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. A Call Agent also may request certain signals to be applied to an endpoint, e.g., dial-tone.

Events and signals are grouped in packages within which they share the same namespace, which we will refer to as event names in the following. A package is a collection of events and signals supported by a particular endpoint-type. For instance, one package may support a certain group of events and signals for analog access lines, and another package may support another group of events and signals for video lines. One or more packages may exist for a given endpoint-type, and each endpoint-type has a default package with which it is associated.

Event names consist of a package name and an event code and, since each package defines a separate namespace, the same event codes may be used in different packages.

³ The list of digits may include other events as well - see Section 3.4.3.1.

⁴ Technically speaking with the “accumulate according to digit map” action.

Package names and event codes are case insensitive strings of letters, digits, and hyphens, with the restriction that hyphens shall never 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. The package name is separated from the event code by a slash (“/”). The package name may be excluded from the event name, in which case the default package name for the endpoint-type in question is assumed. For example, for an analog access line with the line package (package name “L”) being the default package, the following two event names are considered equal:

L/dl dial-tone in the line package for an analog access line.

dl dial-tone in the line package (default) for an analog access line.

This document defines the following packages for the embedded client endpoint-types listed:

Endpoint-Type	Package	Package Name	Default Package
Analog Access Line	Line	L	Yes
Analog Access Line	ADSI	S	No
Video	<i>For further study</i>	<i>For further study</i>	<i>For further study</i>

Additional package names and event codes may be defined by and/or registered with PacketCable™. Any change to the packages defined in this document MUST result in a change of the package name, or a change in the NCS profile version number, or possibly both.

Each package MUST have a package definition, which MUST define the name of the package, and the definition of each event belonging to the package. The event definition MUST include the precise name of the event, i.e., the event code, a plain text definition of the event and, when appropriate, the precise definition of the corresponding signals, for example the exact frequencies of audio signals such as dial-tone or DTMF tones. Events must further specify if they are persistent (e.g., off-hook, see Section 3.3.1) and if they contain auditable event-states (e.g. off-hook, see Section 3.3.8.1). Signals MUST also have their type defined (On/Off, Time-Out, or Brief), and Time-Out signals MUST have a default time-out value defined – see Section 3.3.1, p. 18.

In addition to PacketCable™ packages, implementers MAY gain experience by defining experimental packages. The package name of experimental packages MUST begin with the two characters “x-” or “X-”; PacketCable™ MUST NOT register package names that start with these two characters.

Package names and event codes support one wild-card notation each. The wildcard character “*” (asterisk) can be used to refer to all packages supported by the endpoint in question, and the event code “all” to all events in the package in question. For example

L/all refers to all events in the line package for an analog access line.

*/all for an analog access line; refers to all packages and all events in those packages supported by the endpoint in question.

Consequently, the package name “*” MUST NOT be assigned to a package, and the event code “all” MUST NOT be used in any package.

3.2 SDP Use

The Call Agent uses the MGCP to provide the gateways with the description of connection parameters such as IP addresses, UDP port, and RTP profiles. Except where otherwise noted or implied in this specification, SDP descriptions MUST follow the conventions delineated in the session description protocol (SDP), which is now an IETF-proposed standard documented in *RFC 2327*.

SDP allows for description of multimedia conferences. The NCS profile will only support the setting of audio and video connections using the media types “audio” and “video.” Currently, only “audio” connections have been specified.

3.3 Gateway Control Functions

This section describes the commands of the MGCP in the form of a remote procedure call (RPC) like API, which we will refer to as the media gateway control interface (MGCI). An MGCI function is defined for each MGCP command, where the MGCI function takes and returns the same parameters as the corresponding MGCP command. The functions shown in this section provide a high-level description of the operation of MGCP and describe an example of an RPC-like API that MAY be used for an implementation of MGCP. Although the MGCI API is merely an example API, the semantic behavior defined by MGCI is an integral part of the specification, and all implementations MUST conform to the semantics specified for MGCI. The actual MGCP messages exchanged, including the message formats and encodings used are defined in the protocol section (Section 4). Embedded clients MUST implement those exactly as specified.

The MGCI service consists of connection handling and endpoint handling commands. The following is an overview of the commands:

- 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 on the specified endpoint.
- 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 to a previously established connection.

- The Call Agent can use the DeleteConnection command to delete an existing connection. In some circumstances, the DeleteConnection command also can 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 are generally desirable, e.g., information about the status of the embedded client. 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 the endpoint, or 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. Currently, an endpoint is limited to a specific analog access line within an embedded client.

Connections are grouped into “calls.” 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), “network loopback” (netwloop) or “network continuity test” (netwtest). The “mode” parameter determines if media packets can be sent and/or received on the connection; however, RTCP is unaffected.

Audio signals received from the endpoint will be sent on any connection for that endpoint whose mode is either “send only,” “send/receive,” or “conference.”

Handling of the audio signals received on these connections is also determined by the mode parameters:

- Audio signals received in data packets through connections in “inactive” mode are discarded.
- Audio signals received in data packets through connections in “receive only,” “conference,” or “send/receive” mode are mixed and sent to the endpoint.
- Audio signals originating from the endpoint are transmitted over all the connections whose mode is “send only,” “conference,” or “send/receive.”
- In addition to being sent to the endpoint, audio signals received in data packets through connections in “conference” mode are replicated to all the other connections for the endpoint whose mode is “conference.” The details of this forwarding, e.g., RTP translator or mixer, etc., is outside the scope of this document.

- Audio signals received in data packets through connections in “network loopback” or “network continuity test” mode will be sent back on the connection as described below.

If the mode is set to “network loopback,” the audio signals received from the connection will be echoed back on the same connection. The “network loopback” mode SHOULD simply operate as an RTP packet reflector.

The “network continuity test” mode is used for continuity checking across the IP network. An endpoint-type specific signal is sent to the endpoints over the IP network, and the endpoint is then supposed to echo the signal over the IP network after passing it through the gateway’s internal equipment to verify proper operation. The signal MUST go through internal decoding and re-encoding prior to being passed back. For analog access lines, the signal will be an audio signal, and the signal MUST NOT be passed on to a telephone connected to the analog access line, regardless of the current hook-state of that handset, i.e., on-hook or off-hook

New and existing connections for the endpoint MUST NOT be affected by connections placed in “network loopback” or “network continuity test” mode. However, local resource constraints may limit the number of new connections that can be made.

Please refer to Appendix B for illustrations of mode interactions.

3.3.1 NotificationRequest

The NotificationRequest command is used to request the gateway to send a notification upon the occurrence of specified events in an endpoint. For example, a notification may be requested when tones associated with fax communication are detected on the endpoint. The entity receiving this notification, usually the Call Agent, may then decide that a different type of encoding should be used on the connections bound to this endpoint and instruct the gateway accordingly⁵.

ReturnCode

```
← NotificationRequest(EndpointId
    [, NotifiedEntity]
    [, RequestedEvents]
    , RequestIdentifier
    [, DigitMap]
    [, SignalRequests]
    [, QuarantineHandling]
    [, DetectEvents])
```

⁵ The new instruction would be a ModifyConnection command.

EndpointId is the identifier for the endpoint(s) in the gateway where NotificationRequest executes. The EndpointId follows the rules for endpoint names specified in Section 3.1.1. 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 notification it may trigger. It will be repeated in the corresponding Notify command.

SignalRequests is a parameter that contains the set of signals that the gateway is asked to apply. Unless otherwise specified, signals are applied to the endpoint, however some signals can be applied to a connection instead. The following are examples of signals⁶:

- Ringing,
- Busy tone,
- Call waiting tone,
- Off hook warning tone,
- Ringback tones on a connection

Signals are divided into different types depending upon their behavior:

- On/off (OO). Once applied, these signals last until they are turned off. This can only happen as the result of a new SignalRequests where the signal is 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 the endpoint restarts.
- 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 signal that times-out will generate an operation-complete event (please see Appendix A for further definition of this 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 stopped⁷. 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

⁶ Please refer to Appendix A for a complete list of signals.

⁷ The “Keep signal(s) active” action may override this behavior.

to be notified. The operation-complete event sent to the Call Agent will include the name(s) of the signal(s) that timed out. Time-out signals have a default time-out value defined for them, which may be altered by the provisioning process. A value of zero indicates that the time-out period is infinite.

- **Brief (BR)**. The duration of these signals is 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 will be cancelled. 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 finish playing.

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.*), the media stream **MUST NOT** be forwarded on any connection associated with that endpoint, regardless of the state 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 (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. A given signal **MUST NOT** appear more than once in a SignalRequests.

The currently defined signals can be found in Appendix A.

RequestedEvents is a list of events that the gateway is requested to detect on the endpoint. Examples of events are:

- fax tones,
- modem tones,
- on-hook transition (occurring in classic telephone sets when the user hangs up the handset),
- off-hook transition (occurring in classic telephone sets when the user lifts the handset),
- flash hook (occurring in classic telephone sets when the user briefly presses the hook that holds the handset),
- DTMF digits (or pulse digits).

The currently defined events can be found in Appendix A.

To each event is associated one or more **actions** that define the action that the gateway must take when the event in question occurs. The possible actions are:

- Notify the event immediately, together with the accumulated list of observed events,
- Accumulate the event,
- Accumulate according to Digit Map,
- Ignore the event,
- Keep Signal(s) active,
- Embedded NotificationRequest.

Two sets of requested events will be detected by the endpoint; persistent and non-persistent.

Persistent events are always detected on an endpoint. 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 action⁸. 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 performed⁹. Persistent events are identified as such through their definition – see Appendix A.

Non-persistent events are those events 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 then 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 “Ignore off-hook” was specified, and a new request without any off-hook instructions were received, the default “Notify off-hook” operation then would be restored. A given event **MUST NOT** appear more than once in a RequestedEvents.

More than one action can be specified for an event, although a given action can not appear more than once for a given event. The following matrix specifies the legal combinations of actions:

⁸ Thus the RequestIdentifier will be the RequestIdentifier

⁹ Normally, if a request to look for, e.g., off-hook, is made, the request is only successful if the phone is not already off-hook.

	Notify	Accumulate	Accumulate according to digit map	Ignore	Keep Signal(s) Active	Embedded Notification Request
Notify	-	-	-	-	√	-
Accumulate	-	-	-	-	√	√
Accumulate according to digit map	-	-	-	-	√	-
Ignore	-	-	-	-	√	-
Keep Signal(s) active	√	√	√	√	-	√
Embedded Notification Request	-	√	-	-	√	-

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

When multiple actions are specified, e.g., “Keep signal(s) active” and “Notify,” the individual actions are assumed to occur simultaneously.

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

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 when the Call Agent provides a RequestedEvents parameter with the action “accumulate according to digit map” for that endpoint. The digit map provided is persistent and, therefore, need not be provided whenever a request to “accumulate according to digit map” is made, however Call Agents can provide a digit map at any time. A digit map MUST be provided for the endpoint no later than with the first request to “accumulate according to digit map.” 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).

Each endpoint has a variable called the “current dial string” in which digits are collected for matching with the digit map, as specified in Section 3.1.5, p. 12. Whenever a Notify is sent or a NotificationRequest is to be processed, the “current dial string” is initialized to a null string. The digits to be processed may now either be detected as input, or they may be retrieved from an event input holding area known as the “quarantine buffer” - please see Section 3.4.3.1, p. 45 for further details.

The signals being applied by the SignalRequests is synchronized with the collection of events specified or implied in the RequestedEvents parameter, except if overridden by the

“Keep signal(s) active” action. For example, if the NotificationRequest mandates a “ringing” signal and the event request asks to look for an “off-hook” event, the ringing shall, by default, stop as soon as the gateway detects an off-hook event. If the event request did not ask to look for an “off-hook” event, the ringing would stop anyway since off-hook is a persistent event and therefore implied in the RequestedEvents parameter. The formal definition is that the generation of all “Time Out” signals MUST stop as soon as one of the requested events is detected, unless the “Keep signal(s) active” action is associated to the specified event. In the case of the action “accumulate according to digit map,” the default behavior would be to stop all active time-out signals when the first digit¹⁰ is accumulated—it is irrelevant to this synchronization if the accumulated digit results in a match, mismatch, or partial matching to the digit map.

If it is desired that time-out signal(s) continue when a looked-for event occurs, the “Keep Signal(s) Active” action can be used. This action has the effect of keeping all currently active time-out signal(s) active, thereby negating the default stopping of time-out signals upon the event’s occurrence.

If signal(s) are desired to start when 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. However, the “Embedded NotificationRequest” cannot include another “Embedded NotificationRequest.” When the “Embedded NotificationRequest” is activated, the “current dial string” will be cleared; the list of observed events and the quarantine buffer will be unaffected (see Section 3.4.3.1, p. 45).

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 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.

Finally, 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 signal will still occur by default.

Section 3.4.3.1, p. 45 contains additional details on the semantics of event detection and reporting. The reader is encouraged to study it carefully.

The specific definition of actions that are requested via these SignalRequests (e.g., the duration of and frequency of a DTMF digit) is outside the scope of the core NCS

¹⁰ Digit as defined in digit maps, i.e., including asterisk, timer, etc.

Specification. This definition may vary from location to location and, hence, from gateway to gateway. Consequently, the definitions are provided in event packages, which may be provided outside of the core specification. An initial list of event packages can be found in Appendix A.

The RequestedEvents and SignalRequests generally refer to the same events. 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. There are only a few exceptions to this rule, notably the ADSI display, which can only be signaled but not detected, and the fax and modem tones, which can be detected but can not be signaled. However, we necessarily cannot expect all endpoints to detect all events. The specific events and signals that a given endpoint can detect or perform are determined by the list of event packages that are supported by that endpoint. Each package specifies a list of events and signals that can be detected or applied. A gateway that is requested to detect or to apply an event belonging to a package that is not supported by the specified endpoint MUST return an error (error code 512 or 513 – not equipped to detect event or generate signal). 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).

The Call Agent can send a NotificationRequest whose requested signal list is empty. This has the effect of stopping all active time-out signals. It can do so, for example, when tone generation should stop.

QuarantineHandling is an optional parameter that specifies handling options for the quarantine buffer (see Section 3.4.3.1, p. 45). It allows the Call Agent to specify whether quarantined events should be processed or discarded. If the parameter is absent, the quarantined events MUST be processed.

DetectEvents is an optional parameter that specifies a minimum list of events that the gateway is requested to detect in the “notification” and “lockstep” state. The list is persistent until a new value is specified. Further explanation of this parameter may be found in Section 3.4.3.1, p. 45.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.2 Notifications

Notifications are sent via the Notify command by the gateway when an observed event is to be notified:

```
ReturnCode
  ← Notify(EndpointId
           [, NotifiedEntity])
```

, RequestIdentifier
, ObservedEvents)

EndpointId is the name for the endpoint in the gateway, which is issuing the Notify command, as defined in Section 3.1.1. The identifier **MUST** be a fully qualified endpoint name, including the domain name of the gateway. The local part of the name **MUST NOT** use the wildcard convention.

NotifiedEntity is an optional parameter that identifies the entity to which the notification is sent. 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 notification request that triggered it. Persistent events will be viewed here as if they had been included in the last NotificationRequest. When no NotificationRequest has been received, the RequestIdentifier used will be zero (“0”).

ObservedEvents is a list of events that the gateway detected and accumulated, either by the “accumulate,” “accumulate according to digit map,” or “notify” action. A single notification can report a list of events that will be reported in the order in which they were detected. The list can only contain persistent events and events that were requested in the RequestedEvents parameter of the triggering NotificationRequest. The list will contain the events that were either accumulated (but not notified) or accumulated 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 are 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.”

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.3 CreateConnection

This command is used to create a connection.

```

ReturnCode
, ConnectionId
[, SpecificEndPointId]
, LocalConnectionDescriptor
    ← CreateConnection(CallId
                                , EndpointId
                                [, NotifiedEntity]
                                , LocalConnectionOptions
                                , Mode
                                [, RemoteConnectionDescriptor]
                                [, RequestedEvents]
                                [, RequestIdentifier]
                                [, DigitMap]
                                [, SignalRequests]
                                [, QuarantineHandling]
                                [, DetectEvents])

```

This function creates a connection between two endpoints. A connection is defined by its endpoints and attributes. 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 is, at a minimum, unique within the collection of Call Agents that control the same gateways; connections that belong to the same call share the same call-id. The call-id can be used to identify calls for reporting and accounting purposes.

EndpointId is the identifier for the endpoint in the gateway where CreateConnection executes. The EndpointId can be specified fully by assigning a non-wildcarded value to the parameter EndpointId in the function call or it can be under-specified by using the “anyone” wildcard convention. If the endpoint is under-specified, the endpoint identifier will be assigned by the gateway and its complete value returned in the **SpecificEndPointId** parameter of the response. The “all” wildcard convention MUST not be used.

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

LocalConnectionOptions is a structure that describes the characteristics of the media data connection from the point-of-view of the gateway executing CreateConnection. It instructs the endpoint on the send and, where noted below, receives characteristics of the media connection. The fields contained in LocalConnectionOptions are:

- **Encoding Method.** A literal name for the compression algorithm (encoding/decoding method) used to send and receive media on the connection MUST be specified and

with exactly one value. If the endpoint receives any media on the connection encoded with a different encoding method, it MAY discard it. A list of permissible encoding methods are specified in a separate PacketCable™ document.

- **Packetization Period.** The packetization period in milliseconds, as defined in the SDP standard (RFC 2327), MUST be specified and with exactly one value. The value only pertains to media sent. A list of permissible packetization periods are specified in a separate document.
- **Bandwidth.** The maximum amount of bandwidth in kilobits per second in each direction (send and receive) MAY be specified. If x kilobits is specified, then x kilobits worth of send bandwidth, and x kilobits worth of receive bandwidth may be consumed. However, currently the embedded client device SHOULD ignore any value specified. Details on Quality-of-Service, which are currently TBD and will be provided in both the PacketCable™ Architecture Report and the PacketCable™ Quality-of-Service Specification.
- **Echo Cancellation.** Whether echo cancellation should be used on the line side or not¹¹. The parameter can have the value “on” (when the echo cancellation is requested) or “off” (when it is turned off). The parameter is optional. When the parameter is omitted, the embedded client MUST apply echo cancellation.
- **Type of Service.** Specifies the class of service that will be used for sending media on the connection by encoding the 8-bit type of service value parameter of the IP header as two hexadecimal digits. The parameter is optional. When the parameter is omitted, a default value of five applies. Details on Quality-of-Service, which are currently TBD and will be provided in a separate document may alter this behavior.
- **Silence Suppression.** Whether silence suppression should be used or not in the send direction. The parameter can have the value “on” (when silence is to be suppressed) or “off” (when silence is not to be suppressed). The parameter is optional. When the parameter is omitted, the default is not to use silence suppression.

“The embedded client MUST respond with an error (error code 524 – LocalConnectionOptions inconsistency) if any of the above rules are violated. All of the above mentioned default values can be altered by the provisioning process.

RemoteConnectionDescriptor is the connection descriptor for the remote side of a connection, on the other side of the IP network. It includes the same fields as the LocalConnectionDescriptor (not to be confused with LocalConnectionOptions), i.e., the fields that describe a session according to the SDP standard, however NCS does not support the full SDP standard. Section 4.4, p. 71 details the supported use of SDP in the NCS profile. This parameter may have a null value when the information for the remote

¹¹ Echo cancellation on the packet side is not supported.

end is not known. This occurs because the entity that builds a connection starts by sending a CreateConnection to one of the two gateways involved. For the first CreateConnection issued, there is no information available about the other side of the connection. This information may be provided later via a ModifyConnection call. Apart from the RTP address and port information in a RemoteConnectionDescriptor, all other information is to be considered purely informational and the embedded client MAY ignore it at will. It is purely the responsibility of the Call Agent to ensure that it issues coherent commands to each endpoint to ensure that the codec used by each are the same. When codecs are changed during a call, small periods of time may exist where the endpoints use different codes. As stated above, embedded clients MAY discard any media received that is encoded with a different codec than what is specified in the LocalConnectionOptions for a connection.

Mode indicates the mode of operation for this side of the connection. The options are “send only,” “receive only,” “send/receive,” “conference,” “inactive,” “network loopback” or “network continuity test.” The handling of these modes is specified in the beginning of Section 3.3, p. 17. Some endpoints may not be capable of supporting all modes. If the command specifies a mode that the endpoint does not support, an error MUST be returned (error code 517 – unsupported mode). 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,” or “conference” (error code 527 – missing RemoteConnectionDescriptor).

ConnectionId is a parameter returned by the gateway that uniquely identifies the connection within the context of the endpoint in question.

LocalConnectionDescriptor is a parameter returned by the gateway, which is a session description that contains information about addresses and RTP ports as defined in SDP. It is similar to the RemoteConnectionDescriptor, except that it specifies this side of the connection. Section 4.4, p. 71, details the supported use of SDP in the NCS profile.

After receiving a “CreateConnection” command that does not include a RemoteConnectionDescriptor parameter, a gateway is in an ambiguous situation for the connection in question. Because it has exported a LocalConnectionDescriptor parameter, it potentially can receive packets on that connection. Because it has not yet received the other gateway’s RemoteConnectionDescriptor parameter, it does not know whether the packets it receives have been authorized by the Call Agent. Thus, it must 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 “receive only,” the gateway MUST accept the voice signals received on the connection and transmit them through to the endpoint.
- If the mode was set to “inactive,” the gateway MUST discard the voice signals received on the connection.

- If the mode was set to “network loopback” or “network continuity test” the gateway MUST perform the expected echo or response. The echoed or generated media MUST then be sent to the source of the media received.

Note, that when the endpoint does not have a RemoteConnectionDescriptor for the connection, the connection can by definition not be in any of the modes “send only”, “send/receive”, or “conference”.

The **RequestedEvents**, **RequestIdentifier**, **DigitMap**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are all optional. They can be used by the Call Agent to effectively include a notification request that is executed simultaneously with the creation of the connection. If one or more of these parameters is present, the RequestIdentifier MUST be one of them. Thus, the inclusion of a notification request can be recognized by the presence of a RequestIdentifier. The rest of the parameters may or may not be present. If one of the parameters is not present, it MUST be treated as if it was a normal NotificationRequest with the parameter in question being omitted. This may have the effect of canceling signals and of stop looking for events.

As an example of use, consider a Call Agent that wants to place a call to an embedded client. The Call Agent should:

- ask the embedded client to create a connection, in order to be sure that the user can start speaking as soon as the phone goes off hook,
- ask the embedded client to start ringing,
- ask the embedded client to notify the Call Agent when the phone goes off-hook.

All of the above can be accomplished in a single CreateConnection command by including a notification request with the RequestedEvents parameters for the off-hook event and the SignalRequests parameter for the ringing signal.

When these parameters are present, the creation of the connection and the notification request MUST be synchronized, which means that they are both either accepted or refused. In our example, the CreateConnection must be refused if the gateway does not have sufficient resources or cannot get adequate resources from the local network access. The off-hook notification request must 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. An error would be returned instead (error code 401 – phone off hook), which informs the Call Agent of the glare condition.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.4 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]
    ← ModifyConnection(CallId
                                , EndpointId
                                , ConnectionId
                                [, NotifiedEntity]
                                [, LocalConnectionOptions]
                                [, Mode]
                                [, RemoteConnectionDescriptor]
                                [, RequestedEvents]
                                [, RequestIdentifier]
                                [, DigitMap]
                                [, SignalRequests]
                                [, QuarantineHandling]
                                [, DetectEvents])

```

The parameters used are the same as in the CreateConnection command, with the addition of a **ConnectionId** that uniquely identifies the connection within the endpoint. This parameter is returned by the CreateConnection command together with the local connection descriptor. It uniquely identifies the connection within the context of the endpoint.

The **EndpointId** MUST be a fully qualified endpoint name. The local name MUST NOT use the wildcard convention.

The ModifyConnection command can be used to affect connection parameters, subject to the same rules and constraints as specified for CreateConnection:

- Provide information on the other end of the connection through the **RemoteConnectionDescriptor**.
- Activate or deactivate the connection by changing the **mode** parameter's value. This can occur at any time during the connection, with arbitrary parameter values. An activation can, for example, be set to the "receive only" mode.
- 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.

The command will only return a **LocalConnectionDescriptor** if the local connection parameters, such as RTP ports, are modified. Thus, if, e.g., only the mode of the connection is changed, a LocalConnectionDescriptor will not be returned. If a connection

parameter is omitted, e.g., mode or silence suppression, the value of that parameter will not be changed.

The RTP address information provided in the RemoteConnectionDescriptor specifies the remote RTP address of the receiver of media for the connection. This RTP address information may have been changed by the Call Agent¹². When RTP address information is given to an embedded client for a connection, the embedded client SHOULD only accept media streams (and RTCP) from the RTP address specified as well. Any media streams received from any other addresses SHOULD be discarded. The actual security requirements will be provided in the PacketCable™ Security Specification document and may alter the above behavior.

The **RequestedEvents**, **RequestIdentifier**, **DigitMap**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are optional. The parameters can be used by the Call Agent to include a notification request that is tied to and executed simultaneously with the connection modification. If one or more of these parameters is supplied, then RequestIdentifier MUST be one of them. For example, when a call is accepted, the calling gateway should be instructed to place the connection in “send/receive” mode and to stop providing ringback tones. This can be accomplished in a single ModifyConnection command by including a notification request with the RequestedEvents parameters for the on-hook event, and an empty SignalRequests parameter, to stop the provision of ringback tones.

When these parameters are present, the connection modification and the notification request MUST be synchronized, which means that they are both either accepted or refused.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.5 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
, Connection-parameters
  ← DeleteConnection(CallId
                        , EndpointId
                        , ConnectionId
                        [, NotifiedEntity]
                        [, RequestedEvents]

```

¹² For instance if media needs to traverse a firewall.

[, RequestIdentifier]
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

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

In the general case where a connection has two ends, this command has to be sent to both gateways involved in the connection. After the connection has been deleted, packet network 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.

In response to the DeleteConnection command, the gateway returns a list of parameters that describe the status of the connection. These parameters are:

- **Number of packets sent.** The total number of RTP data packets transmitted by the sender since starting transmission on the connection. 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 Modify command. The value is zero if, e.g., the connection was always set in “receive only” mode.
- **Number of octets sent.** The total number of payload octets (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission on the connection. 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, e.g., the connection was always set in “receive only” mode.
- **Number of packets received.** The total number of RTP data packets received by the sender since starting reception on the connection. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always set in “send only” mode.
- **Number of octets received.** The total number of payload octets (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission on the connection. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always set in “send only” mode.
- **Number of packets lost.** The total number of RTP data 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 are duplicates. 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, e.g., the connection was always set in “send only” mode.

- **Interarrival jitter.** An estimate of the statistical variance of the RTP data packet interarrival time measured in milliseconds and expressed as an unsigned integer. 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, e.g., the connection was always set in “send only” mode.
- **Average transmission delay.** An estimate of the network latency, expressed in milliseconds. 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 and then dividing by the number of RTCP messages that have been received. It should be noted that the correct calculation of this parameter relies on synchronized clocks. Embedded client devices MAY alternatively estimate the average transmission delay by dividing the measured roundtrip time by two.

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

The **NotifiedEntity**, **RequestedEvents**, **RequestIdentifier**, **DigitMap**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are optional. They can be used by the Call Agent to transmit a notification request that is tied to and executed simultaneously with the deletion of the connection. However, if one or more of these parameters are present, RequestIdentifier MUST be one of them. For example, when a user hangs up the phone, the gateway might be instructed to delete the connection and to start looking for an off-hook event. This can be accomplished in a single DeleteConnection command also by transmitting the RequestedEvents parameter for the off-hook event and an empty SignalRequests parameter.

When these parameters are present, the delete connection and the notification request MUST be synchronized, which means that they are both either accepted or refused.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.6 DeleteConnection (From the Embedded Client)

In some circumstances, a gateway may have to clear a connection, for example, because it has lost the resource associated with the connection. The gateway can terminate the connection by using a variant of the DeleteConnection command:

```

ReturnCode
  ← DeleteConnection(CallId,
                      EndpointId,
                      ConnectionId,
                      Reason-code,
                      Connection-parameters)

```

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

The **Reason-code** is a text string starting with a numeric reason-code and optionally followed by a descriptive text string. A list of reason-codes can be found in Section 3.6.

In addition to the **CallId**, **EndpointId**, and **ConnectionId**, the embedded client will also send the connection's parameters, which would have been returned to the Call Agent in response to a DeleteConnection command from the Call Agent. The reason code indicates the cause of the disconnection.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.7 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. The command can be used to delete all connections that relate to a call for an endpoint:

```

ReturnCode
  ← 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. The command does not return any individual statistics or call parameters.

DeleteConnection can also be used by the Call Agent to delete all connections that terminate in a given endpoint:

```

ReturnCode
  ← DeleteConnection(EndpointId)

```

In this form of the DeleteConnection command, Call Agents can take advantage of the hierarchical naming structure of endpoints to delete all the connections that belong to a group of endpoints. In this case, part of the “local endpoint name” component of the EndpointId can be specified using the “all” wildcarding convention, as specified in

Section 3.1.1. The “any of” wildcarding convention MUST NOT be used. The command does not return any individual statistics or call parameters.

After the connection has been deleted, packet network 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.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

3.3.8 Auditing

The MGCP is based upon a centralized call control architecture where a Call Agent acts as the remote controller of client devices that provide telephony interfaces to users and networks. In order to achieve the same or higher levels of availability as the current PSTN, some protocols have implemented mechanisms to periodically “ping” subscribers in order to minimize the time before an individual outage is detected. In this interest, an MGCP-specific auditing mechanism between the embedded clients and the Call Agents in a PacketCable™ system is provided to allow the Call Agent to audit endpoint and connection state and to retrieve protocol-specific capabilities of an endpoint.

Two commands for auditing are defined for the embedded clients:

- **AuditEndPoint:** Used by the Call Agent to determine the status of an endpoint
- **AuditConnection:** Used by the Call Agent to obtain information about a connection.

Network management beyond the capabilities provided by these commands is generally desirable, e.g., information about the status of the embedded client as opposed to individual endpoints. Such capabilities are expected to be supported by the use of the Simple Network Management Protocol (SNMP) and by definition of a MIB for the embedded client, both of which are outside the scope of this specification.

3.3.8.1 AuditEndPoint

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

```
ReturnCode
[, EndPointIdList] |
```

```
ReturnCode
[, RequestedEvents]
[, DigitMap]
[, SignalRequests]
```

```

[, RequestIdentifier]
[, NotifiedEntity]
[, ConnectionIdentifiers]
[, DetectEvents]
[, ObservedEvents]
[, EventStates]
[, LocalConnectionOptions]
    ← AuditEndPoint(EndpointId
                    [, RequestedInfo])

```

The **EndpointId** identifies the endpoint that is being audited. The “any of” wildcard convention **MUST NOT** be used. The “all of” wildcard convention can be used to audit a group of endpoints. If this convention is used, the gateway **MUST** return the list of endpoint identifiers that match the wildcard in the **EndPointIdList** parameter, which is simply a list of SpecificEndpointIds – additional info **MUST NOT** be requested in this case. When the wildcard convention is not used, the (possibly empty) **RequestedInfo** describes the information that is requested for the EndpointId specified. The following endpoint-specific information can then be audited with this command:

RequestedEvents, DigitMap, SignalRequests, RequestIdentifier, NotifiedEntity,
 ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, and
 Capabilities

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

- **RequestedEvents** The current value of RequestedEvents the endpoint is using including the action associated with each event. Persistent events are included in the list.
- **DigitMap** The digit map the endpoint is using currently.
- **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 signals¹³. Time-Out signals that have timed-out, and currently playing Brief signals are not included.
- **RequestIdentifier** The RequestIdentifier for the last NotificationRequest received by the endpoint (includes notification request embedded in connection handling primitives). If no notification request has been received, the value zero will be returned.
- **NotifiedEntity** The current “notified entity” for the endpoint.

¹³ Currently, there should be no pending brief signals.

- **ConnectionIdentifiers** A comma-separated list of ConnectionIdentifiers for all connections that currently exist for the specified endpoint.
- **DetectEvents** The current value of DetectEvents the endpoint is using. Persistent events are included in the list.
- **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. The definition of the individual events will state if the event in question has an auditable state associated with it.
- **Capabilities** The capabilities for the endpoint similar to the LocalConnectionOptions parameter and including event packages and connection modes. 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 sets:
 - **Compression Algorithm** A list of supported codecs. The rest of the parameters 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.
 - **Silence Suppression** Whether silence suppression is supported or not.
 - **Type of Service** Whether type of service is supported or not.
 - **Event Packages** A list of event packages supported. The first event 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.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

If no info was requested and the EndpointId refers to a valid fully-specified EndpointId, the gateway simply returns a successful response (return code 200 – transaction executed normally).

It should be noted, that all of the information returned is merely a snapshot. New commands received, local activity, etc. may alter most of the above. For example the hook-state may change before the Call Agent receives the above information.

3.3.8.2 AuditConnection

Auditing of individual connections on an endpoint can be achieved using the AuditConnection command.

```

ReturnCode
[, CallId]
[, NotifiedEntity]
[, LocalConnectionOptions]
[, Mode]
[, RemoteConnectionDescriptor]
[, LocalConnectionDescriptor]
[, ConnectionParameters]
← AuditConnection(EndpointId
                  , ConnectionId
                  [, RequestedInfo])

```

The **EndpointId** identifies the endpoint that is being audited—wildcards **MUST NOT** be used. 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, ConnectionParameters, RemoteConnectionDescriptor, LocalConnectionDescriptor.

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

- **CallId** The CallId for the call to which the connection belongs.
- **NotifiedEntity** The current “notified entity” for the endpoint.
- **LocalConnectionOptions** The LocalConnectionOptions supplied for the connection.
- **Mode** The current connection mode.
- **ConnectionParameters** Current connection parameters for the connection.
- **LocalConnectionDescriptor** The LocalConnectionDescriptor that the gateway supplied for the connection.

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” method. A restart delay of null for the “restart” method indicates that service has already been restored. This typically will occur after gateway startup/reboot.

Embedded clients SHOULD send a “graceful” or “forced” RestartInProgress message 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, although the Call Agent cannot rely on always receiving such messages. Embedded clients MUST send a “restart” RestartInProgress message with a null delay to their Call Agent when they are back in service according to the restart procedure specified in Section 3.4.3.4 – Call Agents can rely on receiving this message. Also, embedded clients MUST send a “disconnected” RestartInProgress message to their current “notified entity” according to the “disconnected” procedure specified in Section 3.4.3.5. The “restart delay” parameter MUST NOT be used with the “forced” restart method.

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 for each endpoint so, after a reboot, the default Call Agent will be the “notified entity” for each endpoint. Embedded clients 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.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number (see Section 3.5) optionally followed by commentary.

A **NotifiedEntity** may additionally be returned with the response from the Call Agent:

- If the response indicated success (return code 200 – transaction executed), the restart procedure has completed, 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 procedure is not yet complete, and must therefore be initiated again. If a NotifiedEntity parameter was returned, it then specifies the new “notified entity” for the endpoint(s), which must consequently be used when retrying the restart procedure.

3.4 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 may exist when the gateway or the call agent are restarted: the gateway may need to be redirected to a new call agent during “failover” procedures; Similarly, the call agent may need to take special action when the gateway is taken offline, or restarted.

3.4.1 Recaps and Highlights

As mentioned in Section 3.1.4, Call Agents are identified by their domain name, and each endpoint has one, and only one, “notified entity” associated with it at any given point in time. In this section we recap and highlight the areas that are of special importance to reliability and fail-over in MGCP:

- A Call Agent is identified by its domain name, not its network addresses, and several network 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” is initially set to a provisioned value. When commands with a NotifiedEntity parameter is received for the endpoint, including wild-carded endpoint-names, the “notified entity” is set to the value specified. If the “notified entity” for an endpoint is empty or has not been set explicitly¹⁴, the “notified entity” defaults to the source address of the last connection handling command or notification request received for the endpoint. In this case, the Call Agent will thus be identified by its network address, which SHOULD only be done on exceptional basis.
- Responses to commands are always sent to the source address of the command, regardless of the current “notified entity”. When a Notify message needs to be piggy-backed with the response, the datagram is still sent to the source address of the new command received, regardless of the NotifiedEntity for any of the commands.
- When the “notified entity” refers to a domain name that resolves to multiple IP-addresses, endpoints are capable of switching between each of these addresses, however they cannot change the “notified entity” to another domain name on their own. A call agent can however instruct them to switch by providing them with a new “notified entity”.
- If a 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 again, or for another (backup) call agent to contact the affected endpoints with a new “notified entity”.
- When another (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, if so desired. Alternatively, the failed call agent could simply become the backup call agent now.

¹⁴ This could for instance happen by specifying an empty NotifiedEntity parameter.

We should note that handover conflict resolution between separate Call Agent's is not provided - we are relying strictly on the Call Agent's knowing what they are doing and communicating with each other (although AuditEndpoint can be used to learn about the current “notified entity”).

3.4.2 Retransmission, and Detection of Lost Associations

The MGCP protocol is organized as a set of transactions, each of which is composed of a command and a response. The MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response (see Section 4.5), commands are repeated. Gateways **MUST** 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. Recent is here defined by the value T_{hist} that specifies the number of seconds that responses to old transactions must be kept for. The default value for T_{hist} is 30 seconds.

The transaction identifiers of incoming commands are first compared to the transaction identifiers of the recent responses. If a match is found, the gateway does not execute the transaction, but simply repeats the old 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 gateway does not execute the transaction, which is simply ignored - a response will be provided when the execution of the command is complete.

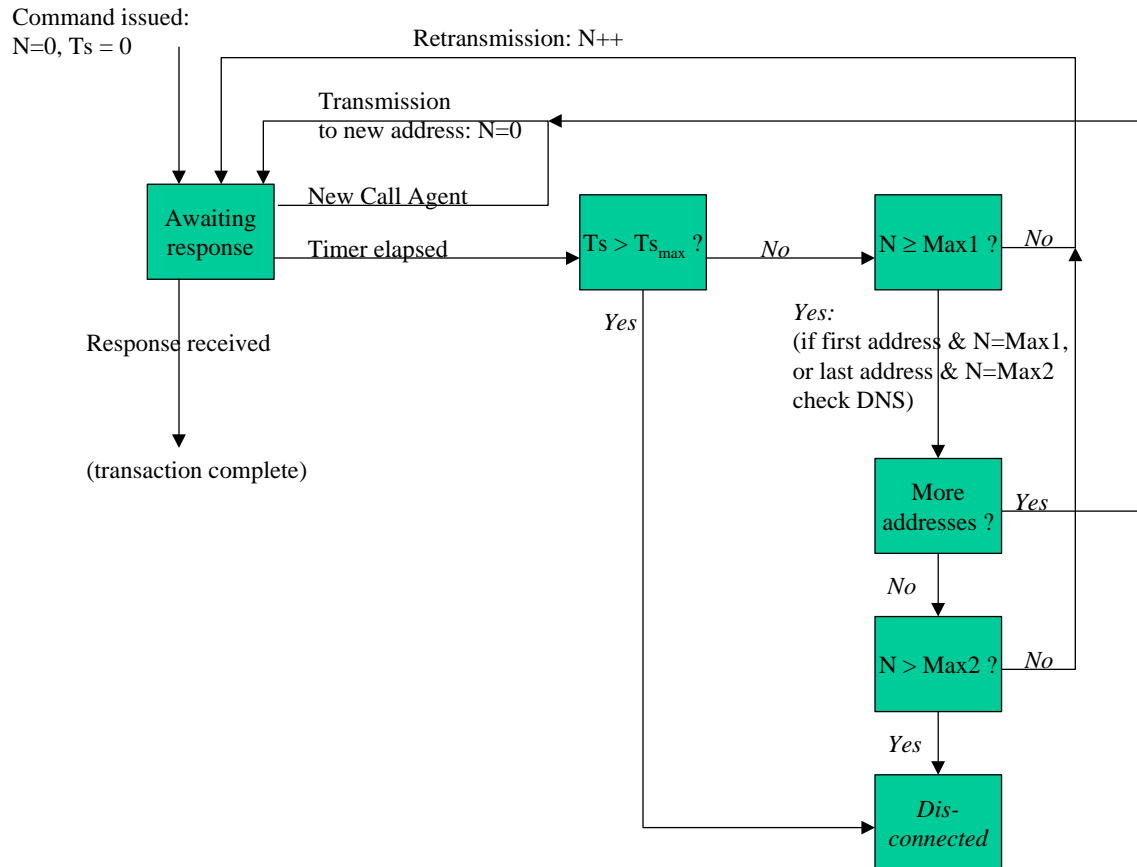
This repetition mechanism is used to guard against four types of possible errors:

- transmission errors, when, e.g., a packet is lost due to noise on a line or congestion in a queue,
- component failure, when, e.g., an interface for a call agent becomes unavailable,
- call agent failure, when, e.g., all interfaces for a 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. In a properly configured system, this loss rate should be very low, typically less than 1% on average. 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 else than a transmission error is occurring. For example, given a uniformly distributed 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 repetitions that is considered excessive should be a function of the prevailing packet loss rate.) When errors are non-uniformly distributed, the consecutive failure probability can become somewhat higher. We should note that the “suspicion threshold,” which we will call “Max1,” is normally lower than the “disconnection threshold,” which we will call “Max2,” and which should be set to a larger value.

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after 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:

- The gateway **MUST** always check for the presence of a new call agent. It can be noticed by:
 - receiving a command where the NotifiedEntity points to a new call agent, or
 - receiving a redirection response pointing to a new call agent.
- If a new Call Agent is detected, the gateway **MUST** direct retransmissions of any outstanding commands 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 $T_{S_{max}}$. If more than $T_{S_{max}}$ time has elapsed, the endpoint becomes disconnected.
- If the number of retransmissions to this Call Agent equals “Max1”, the gateway **MAY** actively query the name server in order to detect the possible change of call agent interfaces, regardless of the Time To Live (TTL) associated with the DNS record.
- The gateway may have learned several IP addresses for the Call Agent. If the number of retransmissions for this IP address is larger than “Max1” and lower than “Max2”, and there are more IP addresses that have not been tried, then the gateway **MUST** direct the retransmissions to the remaining alternate addresses in its local list.
- If there are no more interfaces to try, and the number of retransmissions is Max2, then the gateway **SHOULD** contact the DNS one more time to see if any other interfaces have become available. If not, the endpoint(s) managed by this Call Agent are now disconnected. When an endpoint becomes disconnected, it **MUST** then initiate the “disconnected” procedure as specified in Section 3.4.3.5.



In order to adapt to network load automatically, MGCP specifies exponentially increasing timers (see Section 4.5.2). If the initial time-out 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 retransmissions 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 should be noted, that there is an intimate relationship between $T_{s_{max}}$, $T_{t_{hist}}$, and the maximum propagation delay, $T_{p_{max}}$. Specifically, the following relation MUST be satisfied to prevent retransmitted commands from being executed more than once:

$$T_{t_{hist}} \geq T_{s_{max}} + T_{p_{max}}$$

The default value for $T_{s_{max}}$ is 20 seconds. Thus, if the assumed maximum propagation delay is 10 seconds, then responses to old transactions must 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.

Furthermore, the provisioning process MUST be able to disable one or both of the Max1 and Max2 DNS queries.

3.4.3 Race Conditions

In this section we describe how MGCP deals with race conditions.

First of all, MGCP deals with race conditions through the notion of a “quarantine list” that quarantines events and through explicit detection of desynchronization, e.g., for mismatched hook-state due to glare for an endpoint.

Secondly, 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 by a proper ordering of commands.

Finally, 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 capability are reestablished, many gateways may decide to send RestartInProgress commands simultaneously, which could lead to very unstable operation if not carefully controlled.

3.4.3.1 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”, and the “Detect Events” list.

When the endpoint is initialized, the requested events list and the digit map are empty. After reception of a command, 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 to 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 observed 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 Notify command which will be sent to the “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 detected on the endpoint are stored in a “quarantine” buffer for later processing. The events are, in a sense, “quarantined”. The events detected are the events specified by the union of the RequestedEvents parameter and the most recently received DetectEvents parameter or, in case no DetectEvents parameter has been received, the events that are referred to in the RequestedEvents. Persistent events are detected as well.

The endpoint exits the “notification state” when the response to the Notify command is received¹⁵. The Notify command may be retransmitted in the “notification state”, as specified in Section 3.4.2.

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.

The NCS profile mandates the use of “lockstep mode”, which implies that the gateway MUST receive a new NotificationRequest command after it has sent a Notify command. Until this happens, the endpoint is in a “lockstep state”, and events that occur and are to be detected are simply stored in the quarantine buffer. The events to be quarantined are the same as in the “notification state”. Once the new NotificationRequest is received and executed successfully, the endpoint exits the “lockstep state”.

A gateway can receive at any time a new NotificationRequest command for the endpoint which will also have the effect of taking the endpoint out of the “notification state” assuming the NotificationRequest executes successfully.

When a new NotificationRequest is received in the “notification state”, the gateway shall ensure that the pending Notify is received by the Call Agent prior to a successful response to the new NotificationRequest. It does so by using the “piggy-backing” functionality of the protocol and placing the messages (commands and responses) to be sent in order with the oldest message first. The messages will then be sent in a single packet to the source of the new NotificationRequest, regardless of the source and “notified entity” for the old and new command. The steps involved are the following:

1. the gateway builds a message that carries in a single packet a repetition of the old outstanding Notify command and the response to the new NotificationRequest command.
 2. the endpoint is then taken out of the “notification state” without waiting for the response to the Notify command.
 3. a copy of the outstanding Notify command is kept until a response is received. If a time-out occurs, the Notify will be repeated, in a packet that will also carry a repetition of the response to the NotificationRequest.
- if the packet carrying the response to the NotificationRequest is lost, the Call Agent will retransmit the NotificationRequest. The gateway will reply to this repetition by retransmitting in a single packet the outstanding Notify command and the response to the NotificationRequest – this datagram will be sent to the source of the NotificationRequest

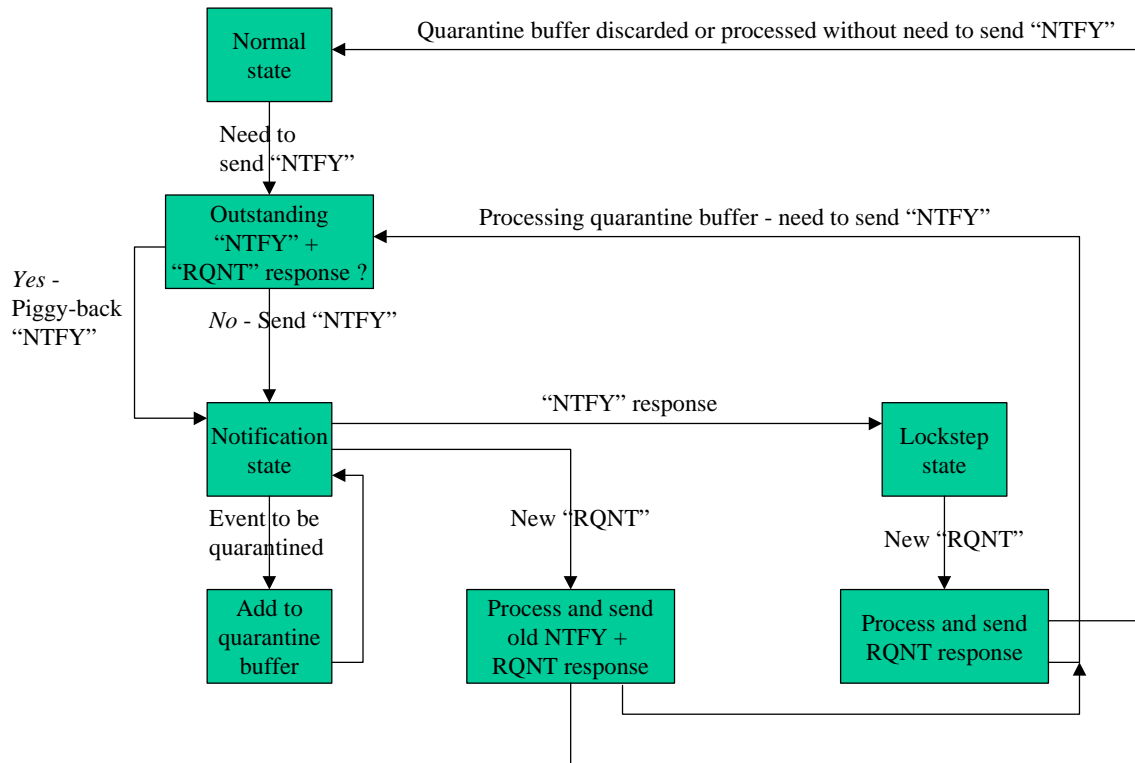
¹⁵ It should be noted, that the Notify action cannot be combined with an Embedded NotificationRequest.

- if the gateway has to transmit a new Notify before a response to the previous Notify is received, it constructs a packet that piggy-backs a repetition of the old Notify, a repetition of the response to the last NotificationRequest, and the new Notify – this datagram will be sent to current “notified entity”.

After receiving a NotificationRequest command, the “requested events” list and “digit map” (if a new one was provided) are replaced by the newly received parameters, and the list of “observed events” and the “current dial string” are reset to a null value. The subsequent behavior is then conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events are to be discarded, in which case all quarantined events are discarded. If the parameter specifies that the quarantined events should be processed, the gateway will start processing the list of quarantined 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 triggers 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” up until and including the triggering event, leaving the unprocessed events in the quarantine buffer. The endpoint then enters the “notification state” again.

The above procedure applies to all forms of notification requests, regardless of whether they are part of a connection handling command or provided as a NotificationRequest command. Connection handling commands that do not include a notification request are neither affected by nor do they affect the above procedure.

The diagram below illustrates the procedure specified above assuming all transactions execute successfully:



Call Agents SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggy-backing mechanism¹⁶.

3.4.3.2 Explicit detection

A key element of the state of several endpoints is the position of the hook. Although hook-state changing events are persistent in NCS, race conditions may still occur, for example when the user decides to go off-hook while the Call Agent is in the process of requesting the gateway to look for off-hook events and perhaps apply a ringing signal (the “glare” condition well known in telephony).

To avoid this race condition, the gateway MUST check the condition of the endpoint before responding to a NotificationRequest. Specifically, 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).

¹⁶ Vendors that choose not to follow this recommendation should examine Call Agent failure scenarios carefully.

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 definition of 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, where as 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 requested events 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 will simply continue as if the command had never been received. although an error is returned. 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 can 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.

3.4.3.3 Ordering of commands, and treatment of disorder

MGCP does not mandate that the underlying transport protocol guarantees the sequencing of commands sent 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 to the call agent after the transmission of a new Notification Request command,
- If a new NotificationRequest is transmitted before a response to a previous one is received, there is no guarantee that the previous one will not be received in second position.

Call Agents and gateways that want to guarantee consistent operation of the endpoints can use the rules specified:

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 MUST be ignored, and an error returned (error code 515 – incorrect connection-id).
4. On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter is used to correlate Notify commands with the triggering NotificationRequest.
5. In some cases, an implicitly or explicitly wild-carded 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 a response to the wild-carded DeleteConnection command is received.
6. When commands are embedded within each other, sequencing requirements for all commands MUST be adhered to. For example a CreateConnection command with a notification request in it must adhere to the sequencing requirements for CreateConnection and NotificationRequest at the same time.
7. AuditEndpoint and AuditConnection is not subject to any sequencing.
8. RestartInProgress must always be the first command sent by an endpoint as defined by the restart procedure (see Section 3.4.3.4). Any other command or response must be delivered after this RestartInProgress command (piggy-backing allowed).
9. When multiple messages are piggy-backed in a single packet, the messages are always processed in order.

Those of the above rules that specify gateway behavior MUST be adhered to by embedded clients, however the embedded client MUST NOT make any assumptions as to whether Call Agents follow the rules or not. Consequently gateways MUST always respond to commands, regardless of whether they adhere to the above rules or not.

3.4.3.4 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 MUST be followed:

1. When a gateway is powered on, it initiates a restart timer to a random value, uniformly distributed between 0 and a provisionable maximum waiting delay (MWD), e.g., 360 seconds (see below). Care MUST be taken to avoid synchronicity

of the random number generation between multiple gateways that would use the same algorithm.

2. The gateway then waits 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. A pre-existing off-hook condition results in the generation of an off-hook event.
3. When the restart timer elapses, when a command is received, or when an activity or pre-existing off-hook condition is detected, the gateway initiates the restart procedure.

The restart procedure simply states that the endpoint **MUST** send a RestartInProgress command to the Call Agent informing it about the restart and furthermore guarantee that the first message (command or response) that the Call Agent sees from this endpoint **MUST** be this RestartInProgress command. The endpoint **MUST** take full advantage of piggy-backing in achieving this. For example, if an off-hook activity occurs prior to the restart timer expiring, a packet containing the RestartInProgress command, and with a piggy-backed Notify command for the off-hook event will be generated. In the case where the restart timer expires without any other activity, the gateway simply sends a RestartInProgress message.

It is expected that 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.

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, on 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 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 suggests that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, embedded clients **MUST** use a default value of 600 seconds for MWD.

3.4.3.5 Disconnected Endpoints

In addition to the restart procedure, embedded clients also have a “disconnected” procedure, which is initiated when an endpoint becomes “disconnected” as described in

Section 3.4.2. It should here be noted, that endpoints can only become disconnected when they attempt to communicate with the Call Agent. The following steps are followed by an endpoint that becomes “disconnected”:

1. A “disconnected” timer is initialized to a random value, uniformly distributed between 0 and a provisionable “disconnected” initial waiting delay (Td_{init}), 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 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, when a command is received, or when a local user activity is detected, the gateway initiates the “disconnected” procedure for the endpoint. In the case of local user activity, a provisionable “disconnected” minimum waiting delay (Td_{min}) must furthermore have elapsed since the gateway became disconnected or the last time it initiated the “disconnected” procedure in order to limit the rate at which the procedure is performed.
4. If the “disconnected” procedure still left the endpoint disconnected, the “disconnected” timer is then doubled, subject to a provisionable “disconnected” maximum waiting delay (Td_{max}), e.g., 600 seconds, and the gateway proceeds with step 2 again.

The “disconnected” procedure is similar to the restart procedure in that it now simply states that the endpoint **MUST** send a RestartInProgress command to the Call Agent informing it that the endpoint was disconnected and furthermore guarantee that the first message (command or response) that the Call Agent now sees from this endpoint **MUST** be this RestartInProgress command. The endpoint **MUST** take full advantage of piggy-backing in achieving this. The Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint.

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 Td_{init} is 15 seconds, the default value for Td_{min} , is 15 seconds, and the default value for Td_{max} is 600 seconds.

3.5 Return Codes and Error Codes

All MGCP commands receive a response. The response carries a return code that indicates the status of the command. The return code is an integer number, for which three value ranges have been defined:

- 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.

The values that have been defined are listed in the following table:

Code	Meaning
200	The requested transaction was executed normally.
250	The connection(s) was deleted.
400	The transaction could not be executed, due to a transient error.
401	The phone is already off hook.
402	The phone is already on hook.
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.
502	The transaction could not be executed because the endpoint does not have sufficient resources.
510	The transaction could not be executed because a protocol error was detected.
511	The transaction could not be executed because the command contained an unrecognized extension.
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.
517	Unsupported or invalid mode.
518	Unsupported or unknown package.
519	Endpoint does not have a digit map.
520	The transaction could not be executed because the endpoint is "restarting."
521	Endpoint redirected to another Call Agent.
522	No such event or signal
523	Unknown action or illegal combination of actions.
524	Internal inconsistency in LocalConnectionOptions
525	Unknown extension in LocalConnectionOptions
526	Insufficient bandwidth
527	Missing RemoteConnectionDescriptor
528	Incompatible protocol version

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

Code	Meaning
900	Endpoint malfunctioning
901	Endpoint taken out of service
902	Loss of lower layer connectivity (e.g., downstream sync)

4 Media Gateway Control Protocol

The 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 eight types of commands:

- CreateConnection
- ModifyConnection
- DeleteConnection
- NotificationRequest
- Notify
- AuditEndpoint
- AuditConnection
- RestartInProgress

The first four commands are sent by the Call Agent to a gateway. The Notify command is sent by the gateway to the Call Agent. The gateway can also send a DeleteConnection as defined in Section 3.3.6. The Call Agent can send either of the Audit commands to the gateway and, finally, the gateway can send a RestartInProgress command to the Call Agent.

4.1 General Description

All commands are composed of a Command header, which for some commands may be followed by a session description.

All responses are composed of a Response header, which for some commands may be followed by a session description.

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 headers are separated from the session description by an empty line.

MGCP uses a transaction identifier with a value between 1 and 999999999 to correlate commands and responses. The transaction identifier is encoded as a component of the command header and is repeated as a component of the response header.

4.2 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 parameter lines composed of a parameter name followed by a parameter value.

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

4.2.1 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 should 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. Embedded clients **SHOULD** use exactly one ASCII space separator, however they **MUST** be able to parse messages with additional white space characters.

4.2.1.1 Requested Verb Coding

Requested verbs are encoded as four letter upper- and/or lower-case ASCII codes (comparisons **MUST** be case insensitive) as defined in the following table:

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

RestartInProgress	RSIP
-------------------	------

New verbs may be defined in future versions of the protocol. It may be necessary, for experimental purposes, to use new verbs before they are sanctioned in a published version of this protocol. Experimental verbs should be identified by a four-letter code starting with the letter X (e.g., XPER).

4.2.1.2 Transaction Identifiers

Transaction identifiers are used to correlate commands and responses.

An embedded client 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 embedded client **MUST** be unique for the maximum lifetime of the transactions within the collection of Call Agents that control that embedded client (see Section 4.5). Thus, regardless of the sending Call Agent, embedded clients 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 embedded client **MUST** be unique for the maximum lifetime of the transactions (see Section 4.5) regardless of which Call Agent the command is sent to. Thus, a Call Agent can always detect a duplicate transaction from an embedded client 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 999999999. 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.

4.2.1.3 Endpoint, Call Agent and NotifiedEntity Name Coding

The endpoint names and Call Agent names are encoded as e-mail addresses, as defined in *RFC 821*. In these addresses, the domain name identifies the system where the endpoint is attached, while the left side identifies a specific endpoint on that system. Both components **MUST** be case insensitive.

Examples of such names are:

aaln/1@ncs2.whatever.net	Analog access line 1 in the embedded client
--------------------------	---

	ncs2 in the “Whatever” network.
Call-agent@ca.whatever.net	Call Agent for the “whatever” network.

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

Call-agent@ca.whatever.net:5234

In case the port number is omitted, the default MGCP port (2427) will be used. Additional detail on endpoint names can be found in Section 3.1.1, p. 7.

4.2.1.4 Protocol Version Coding

The protocol version is coded as the keyword “MGCP” followed by a white space and the version number, which again is followed by the profile name “NCS” and a profile version number. The version numbers are composed of a major version number, a dot, and a minor version number. The major and minor version numbers are coded as decimal numbers. The profile version number defined by this specification is 1.0

The protocol version for this specification MUST be encoded as:

MGCP 0.1 NCS 1.0

The “NCS 1.0” portion signals that this is the NCS 1.0 profile of MGCP 0.1.

4.2.2 Parameter Lines

Parameter lines are composed of a parameter name, which in most cases is composed of a single upper-case character, followed by a colon, a white space, and the parameter value. Parameter names and values are still case-insensitive though. The parameters that can be present in commands are defined in the following table:

Parameter name	Code	Parameter value
CallId	C	Hexadecimal string, at most 32 characters.
ConnectionId	I	Hexadecimal string, at most 32 characters.
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.whatever.net:5234
RequestIdentifier	X	See description
LocalConnectionOptions	L	See description.
Connection Mode	M	See description.
RequestedEvents	R	See description.
SignalRequests	S	See description.
DigitMap	D	A text encoding of a digit map.
ObservedEvents	O	See description.
ConnectionParameters	P	See description.
ReasonCode	E	See description.
SpecificEndPointId	Z	An identifier, in RFC 821 format, composed of an arbitrary string, optionally followed by an “@” followed by the domain name of the embedded client to which this endpoint is attached.
RequestedInfo	F	See description
QuarantineHandling	Q	See description
DetectEvents	T	See description
EventStates	ES	See description
RestartMethod	RM	See description
RestartDelay	RD	A number of seconds encoded as a decimal number.
Capabilities	A	See 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:

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
CallId	M	M	O	F	F	F	F	F
ConnectionId	F	M	O	F	F	F	M	F
RequestIdentifier	O	O	O	M	M	F	F	F
LocalConnectionOptions	M	O	F	F	F	F	F	F
Connection Mode	M	O	F	F	F	F	F	F
RequestedEvents	O	O	O	O*	F	F	F	F
SignalRequests	O	O	O	O*	F	F	F	F
NotifiedEntity	O	O	O	O	O	F	F	F
ReasonCode	F	F	O	F	F	F	F	F

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ObservedEvents	F	F	F	F	M	F	F	F
DigitMap	O	O	O	O	F	F	F	F
Connection parameters	F	F	O	F	F	F	F	F
Specific Endpoint Id	F	F	F	F	F	F	F	F
RequestedInfo	F	F	F	F	F	O	O	F
QuarantineHandling	O	O	O	O	F	F	F	F
DetectEvents	O	O	O	O	F	F	F	F
EventStates	F	F	F	F	F	F	F	F
RestartMethod	F	F	F	F	F	F	F	M
RestartDelay	F	F	F	F	F	F	F	O
Capabilities	F	F	F	F	F	F	F	F
RemoteConnectionDescriptor	O	O	F	F	F	F	F	F

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

Embedded clients and Call Agents SHOULD always provide mandatory parameters before optional ones, however embedded clients MUST NOT fail if this recommendation is not followed.

If implementers need to experiment with new parameters, for example when developing a new MGCP application, they should identify these parameters by names that begin with the string “X-” or “X+”, such as for example:

X-FlowerOfTheDay: Daisy

Parameter names that start with “X+” are mandatory parameter extensions. A gateway that receives a mandatory parameter extension that it cannot understand should refuse to execute the command. It should respond with an error (error code 511 - unrecognized extension).

Parameter names that start with “X-” are non critical parameter extensions. A gateway that receives a non critical parameter extension that it cannot understand can safely ignore that parameter.

It should be noted that experimental verbs are of the form *XABC*, whereas experimental parameters are of the form *X-ABC*.

If a parameter line is received with a forbidden parameter, or any other formatting error, the receiving entity should respond with the most specific error code for the error in question. The least specific error code is 510 – protocol error. Commentary text can always be provided.

4.2.2.1 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest that triggered it. A RequestIdentifier is a hexadecimal string, at most 32 characters. The string “0” is reserved for reporting of persistent events in the case where no NotificationRequest has been received yet (see section 3.3.2, p. 24).

4.2.2.2 Local Connection Options

The local connection options describe the operational parameters that the Call Agents instructs the gateway to use for a connection. These parameters are:

- The packetization period in milliseconds, encoded as the keyword “p” followed by a colon and a decimal number.
- The literal name of the compression algorithm, encoded as the keyword “a” followed by a colon and a character string.
- The bandwidth in kilobits per second (1000 bits per second), encoded as the keyword “b” followed by a colon and a decimal number.
- The echo-cancellation parameter, encoded as the keyword “e” followed by a colon and the value “on” or “off”.
- The type of service parameter, encoded as the keyword “t” followed by a colon and the value encoded as two hexadecimal digits.
- The silence suppression parameter, encoded as the keyword “s” followed by a colon and the value “on” or “off”.

“ ” When several parameters are present, the values are separated by a comma. It is considered an error to include a parameter without a value (error code 524 – LocalConnectionOptions inconsistency).

Examples of local connection options are:

L: p:10, a:PCMU

L: p:10, a:PCMU, e:off, t:1, s:on

L: p:30, b:8, a:G723, e:on, t:05, s:off

4.2.2.3 Capabilities

Capabilities inform the Call Agent about its capabilities when audited. The encoding of capabilities is based on the Local Connection Options encoding for the parameters that are common to both. In addition, capabilities it can also contain a list of supported packages, and a list of supported modes when used to describe capabilities.

The parameters used are:

- The packetization period in milliseconds, encoded as the keyword “p” followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen.
- The literal name of the compression algorithm, encoded as the keyword “a” followed by a colon and a character string. A list of values may be specified in which case the values will be separated by a semicolon.
- The bandwidth in kilobits per second (1000 bits per second), encoded as the keyword “b” followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen.
- The echo-cancellation parameter, encoded as the keyword “e” followed by a colon and the value “on” if echo cancellation is supported, “off” otherwise.
- The type of service parameter, encoded as the keyword “t” followed by a colon and the value “0” if type of service is not supported, all other values indicated support for type of service.
- The silence suppression parameter, encoded as the keyword “s” followed by a colon and the value “on” if silence suppression is supported, “off” otherwise.
- “ ” The event packages supported by this endpoint encoded as the keyword “v” followed by a colon and then a semicolon-separated list of package names supported. The first value specified will be the default package for the endpoint.
- The connection modes supported by this endpoint encoded as the keyword “m” followed by a colon and a semicolon-separated list of connection modes supported as defined in Section 4.2.2.6.

When several parameters are present, the values are separated by a comma.

Examples of capabilities are:

```
A: a:PCMU;G723, p:10-100, e:on, s:off, v:L;S,
m:sendonly;recvonly;sendrecv;inactive
```

```
A: a:G723; p:30-90, e:on, s:on, v:L;S,
m:sendonly;recvonly;sendrecv;inactive;confrnce
```

4.2.2.4 Connection Parameters

Connection parameters are encoded as a string of type and value pairs, where the type is a two-letter identifier of the parameter, and the value a decimal integer. Types are separated from values by an “=” sign. Parameters are separated from each other by a comma.

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 not received on the connection, as deduced from gaps in the 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.

An example of a connection parameter encoding is:

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

4.2.2.5 Reason Codes

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

900 Endpoint malfunctioning

A list of reason-codes can be found in Section 3.6.

4.2.2.6 Connection Mode

The connection mode describes the connection's operation mode. 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 send and receive packets according to conference mode.
M: inactive	The gateway should neither send nor receive packets.
M: netwloop	The gateway should place the endpoint in Network Loopback mode.
M: netwtst	The gateway should place the endpoint in Network Continuity Test mode.

4.2.2.7 Event/Signal Name Coding

Event/signal names are composed of an optional package name, separated by a slash (/) from the name of the actual event. Event names are used in the RequestedEvents, SignalRequests, DetectEvents, ObservedEvents, and EventStates parameters. Each event is identified by an event code. These ASCII encodings are not case sensitive. Values such as “hu,” “Hu,” “HU” or “hU” should be considered equal.

Each signal has one of the following signal-types associated with it (see section 3.3.1, p. 18):

- On/Off (OO),
- Time-out (TO),
- Brief (BR).

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 on:

vmwi(+), vmwi

The following are valid examples of event names:

L/hu	On-hook transition, in the line package
L/0	Digit 0 in the Line package
hf	flash-hook, assuming that the line package is the default package for the endpoint.

In addition, the range and wildcard notation of events can be used, instead of individual names, in the RequestedEvents and DetectEvents (but not SignalRequests or EventStates)

L/[0-9]	Digits 0 to 9 in the Line package
L/X	Digits 0 to 9 in the Line package
hf	Flash-hook, assuming that the line package is a default package for the end point.
[0-9*#A-D]	All digits and letters in the Line package (default for endpoint).
L/all	All events in the Line package.

An initial set of event packages for embedded clients can be found in Appendix A.

4.2.2.8 RequestedEvents

The RequestedEvents parameter provides the list of events that have been requested. The currently defined event codes are described in Appendix A.

Each event can be qualified by a requested action, or by a list of actions. Not all actions can be combined – please refer to Section 3.3.1 for valid combinations. 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
Accumulate according to digit map	D
Ignore	I
Keep Signal(s) active	K
Embedded NotificationRequest	E

If a digit map is not provided when the “accumulate according to digit map” action is specified, the endpoint simply uses its current digit map. If the endpoint does not have any digit maps currently, an error must be returned (error code 519 – no digit map).

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 discarded, except for persistent events.

The digit-map action can only be specified for the digits, letters, and timers.

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

R: hu(N), hf(N) Notify on-hook, notify hook-flash.

R: hu(N), [0-9#T](D) Notify on-hook, accumulate digits according to digit map.

The embedded NotificationRequest follows the format:

E (R(<RequestedEvents>), D(<Digit Map>), S(<SignalRequest>))

with each of R, D, and S being optional and possibly supplied in another order. The following example illustrates the use of Embedded NotificationRequest:

R: hd(A, E(S(dl), R(oc(N), [0-9#T](D)), D(1xxxxxxxxx|9011x.T))) On off-hook, accumulate the event, provide dial-tone and start accumulating digits

according to the digit map supplied. Stop dial-tone when the first digit is input, or, if no digit is input before the dial-tone times out, Notify the operation complete. Otherwise, notify the off-hook and collected digits when a match, mismatch, or inter-digit timeout has occurred. It should be noted, that since on-hook is a persistent event, it will still be notified although it has not been specified here.

4.2.2.9 SignalRequests

The SignalRequests parameter provides the name of the signals that have been requested. The currently defined signals can be found in Appendix A. A given signal can only appear once in the list, and all signals will, by definition, be applied at the same time.

Two signals - caller-id and ADSI display - can be qualified by additional parameters, which in this case is the data that should be displayed.

These parameters will be enclosed within parenthesis, as in (assuming “Line” is the default package):

S: S/adsi(123456 Your friend)

S: ci(10/14/17:26, 555 1212, CableLabs)

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

S: S/adsi(123456 Your friend), rg

4.2.2.10 ObservedEvents

The observed events parameters provide the list of events that have been observed. The event codes are the same as those used in the NotificationRequest. Examples of observed events are:

O: hu

O: 8, 2, 9, 5, 5, 5, 5, T

O: hf, hf, hu

O: 8, 2, 9, 5, mt, 5, 5, 5, T

Events that have been accumulated according to digit map, are reported as individual events in the order they were detected. Other events may be mixed in between them. It should be noted that if the “current dial string” is non-empty with a partial match, and another event occurs that results in a Notify message being generated, the partially matched “current dial string” will be included in the list of observed events, and the “current dial string” will then be cleared – please refer to Section 3.4.3.1 for details.

4.2.2.11 RequestedInfo

The RequestedInfo parameter contains a comma separated list of parameter codes, as defined in the “Parameter lines” section – Section 3.3.8 lists the parameters that can be audited. The following values are supported as well:

RequestedInfo Parameter	Code
LocalConnectionDescriptor	LC
RemoteConnectionDescriptor	RC

For example, if one wants to audit the value of the NotifiedEntity, RequestIdentifier, RequestedEvents, SignalRequests, DigitMap, DetectEvents, EventStates, LocalConnectionDescriptor, and RemoteConnectionDescriptor parameters, the value of the RequestedInfo parameter will be:

F: N, X, R, S, D, T, ES, LC, RC

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

F: A

4.2.2.12 QuarantineHandling

The quarantine handling parameter contains the keyword “process” or “discard” to indicate the treatment of quarantined events, e.g.:

Q: process

4.2.2.13 DetectEvents

The DetectEvents parameter is encoded as a comma separated list of events, such as for example:

T: hu,hd,hf,[0-9#*]

It should be noted, that no actions can be associated with the events.

4.2.2.14 EventStates

The EventStates parameter is encoded as a comma separated list of events, such as for example:

ES: hu

It should be noted, that no actions can be associated with the events.

4.2.2.15 RestartMethod

The RestartMethod parameter is encoded as one of the keywords “graceful”, “forced”, “restart”, or “disconnected”, as for example:

RM: restart

4.3 Response Header Formats

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

The response line starts with the response code, which is a three-digit numeric value. The code is followed by a white space, the transaction identifier, and optional commentary preceded by a white space, e.g.:

200 1201 OK

The following table summarizes the response parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response assuming the command succeeded. The reader should still study the individual command definitions though as this table only provides summary information. The letter M stands for mandatory, O for optional and F for forbidden.

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
CallId	F	F	F	F	F	F	O	F
ConnectionId	M	F	F	F	F	O	F	F
RequestIdentifier	F	F	F	F	F	O	F	F
LocalConnectionOptions	F	F	F	F	F	O	O	F
Connection Mode	F	F	F	F	F	F	O	F
RequestedEvents	F	F	F	F	F	O	F	F
SignalRequests	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	O	O	O
ReasonCode	F	F	F	F	F	F	F	F
ObservedEvents	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	O	F	F
ConnectionParameters	F	F	O	F	F	F	O	F
Specific Endpoint ID	O	F	F	F	F	O	F	F
RequestedInfo	F	F	F	F	F	F	F	F
QuarantineHandling	F	F	F	F	F	F	F	F
DetectEvents	F	F	F	F	F	O	F	F
EventStates	F	F	F	F	F	O	F	F
RestartMethod	F	F	F	F	F	F	F	F
RestartDelay	F	F	F	F	F	F	F	F

Capabilities	F	F	F	F	F	O	F	F
LocalConnection Descriptor	M	O	F	F	F	F	O	F
RemoteConnection Descriptor	F	F	F	F	F	F	O	F

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

4.3.1 CreateConnection

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 in section 4.4, p. 71. It is separated from the response header by an empty line, e.g.:

```
200 1204 OK
I: FDE234C8

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

4.3.2 ModifyConnection

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 in section 4.4, p. 71. It is separated from the response header by an empty line.

```
200 1207 OK

v=0
c=IN IP4 128.96.41.1
m=audio 3456 RTP/AVP 0
```

4.3.3 DeleteConnection

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in Section 4.2.2.4, p. 62.

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

4.3.4 NotificationRequest

A NotificationRequest response does not include any additional response parameters.

4.3.5 Notify

A Notify response does not include any additional response parameters.

4.3.6 AuditEndpoint

In the case of an 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. 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
```

or:

```
200 1200 OK
A: a:PCMU;G.726-32, p:10-100, e:on, s:off, t:1, v:L,
  m:sendonly;recvonly;sendrecv;inactive
A: a:G723; p:30-90, e:on, s:on, t:1, v:L,
  m:sendonly;recvonly;sendrecv;inactive;confrnce
```

4.3.7 AuditConnection

In the case of an 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;G.726-32
M: sendrecv
P: PS=622, OS=31172, PR=390, OR=22561, PL=5, JI=29,
LA=50
```

```
v=0
c=IN IP4 128.96.63.25
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.

4.3.8 RestartInProgress

The response to a RestartInProgress may include the name of another Call Agent to contact, for instance when the Call Agent redirects the endpoint to another Call Agent as in:

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

4.4 Session Description Encoding

The session description is encoded in conformance with the session description protocol (SDP), however, embedded clients may make certain simplifying assumptions about the session description as specified in the following. It should be noted, that session descriptions are case sensitive per RFC 2327.

SDP usage depends on the type of session, as specified in the “media” parameter:

- If the media is set to “audio,” the session description is for an audio service,
- If the media is set to “video,” the session description is for a video service,

For an audio service, the gateway will consider the information provided in SDP for the “audio” media, and for a video service the gateway will consider the information provided in SDP for the “video” media.

4.4.1 SDP Audio Service Use

In a telephony-only gateway, we only have to describe sessions that use exactly one media, audio. The parameters of SDP that are relevant for the telephony application are specified below. Embedded client MUST support session descriptions that conform to these rules.

At the session description level:

- The protocol version field. Embedded clients must support the protocol version field as defined in SDP (RFC2327) and further specified here:

v=0

- The IP address of the remote gateway (in commands), or of the local gateway (in responses), encoded as an SDP “connection data” parameter. This parameter specifies the IP address that will be used to exchange RTP packets.

Embedded clients must support connection data fields as defined in SDP (RFC2327) and further specified here:

c = <network type> <address type> <connection address>

where:

<network type> will be “IN”

<address type> will be “IP4”

<connection address> will be an IPv4 address in dotted decimal form only, e.g., “128.2.3.4”

thus an example could be:

```
c=IN IP4 128.2.3.4
```

Embedded clients will not support specification of the connection data field at the media description level – it must be supplied at the session description level.

For audio media:

- Media description field (m) specifying the audio media, the transport port used for receiving RTP packets by the remote gateway (commands) or by the local gateway (responses), the RTP/AVP transport, and the list of formats that the gateway will accept:

Embedded clients must support media description fields as defined in SDP and further specified here:

```
m = <media> <port> <transport> <fmt list>
```

where:

<media> will be “audio”

<port> is a port number between 1024 and 65535

<transport> will be “RTP/AVP”

<fmt list> is as defined in SDP

thus an example could be:

```
m=audio 49230 RTP/AVP 0
```

which would define PCM (i.e., G.711) u-law.

The actual set of codecs that must be supported is defined in the PacketCable™ Codec Specification, however, at a minimum PacketCable™ embedded clients will support the following:

- Default high bandwidth codec: G.711, u-law (defined as payload type 0 in RFC 1890)
- Low bandwidth codec: TBD
- Wideband codec: TBD
- Optionally, RTPMAP attributes defining the encoding of dynamic audio formats.

Embedded clients must support dynamic payload types in the form of RTPMAP attributes as defined in SDP (*RFC 2327*).

- Optionally, a packetization period (packet time) attribute (ptime) defining the duration of the packet.

Embedded clients must support specification of the packetization period as defined in SDP and further specified here:

```
a=ptime:<packet time>
```

where:

<packet time> is a positive integer that specifies the recommended packetization period per media packet sent.

- Optionally, an attribute defining the connection type (sendonly, recvonly, sendrecv, inactive, conference, etc.).

Embedded clients can safely ignore this SDP parameter.

4.4.2 SDP Video Service Use

Details on SDP use for video service will be provided in a future version of this document.

4.5 Transmission Over UDP

4.5.1 Reliable Message Delivery

MGCP messages are transmitted over UDP. Commands are sent to one of the IP addresses defined in the Domain Name System (DNS) for the specified endpoint or Call Agent. The responses are sent back to the source address of the command. However, it should be noted that the response may, in fact, come from another IP address than the one to which the command was sent.

When no port is provisioned for the endpoint¹⁷, the commands should be sent to the default MGCP port, 2427.

MGCP messages, carried over UDP, may be subject to losses. In the absence of a timely response, commands are repeated. MGCP entities are expected to keep, in memory, a list of the responses sent to recent transactions, i.e., a list of all the responses sent over the last T_{hist} seconds, as well as a list of the transactions that are being executed currently. Transaction identifiers of incoming commands are compared to 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 no match is found, the MGCP entity examines the list of currently executing transactions. If a match is found, the MGCP entity will not execute the transaction, which is simply ignored.

It is the responsibility of the requesting entity to provide suitable timeouts for all outstanding commands and to retry commands when timeouts have been exceeded. A retransmission strategy is specified in Section 4.5.2.

Furthermore, when repeated commands fail to get a response, the destination entity is assumed to be unavailable. It is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending connections as specified in Section 3.4.

4.5.2 Retransmission Strategy

This specification avoids specifying any static values for the retransmission timers since these values are typically network-dependent. Normally, the retransmission timers should estimate the timer by measuring the time spent between sending a command and the return of a response. Embedded clients **MUST** at a minimum implement a retransmission strategy using exponential back-off with configurable initial and maximum retransmission timer values.

Embedded clients **SHOULD** use the algorithm implemented in TCP-IP, which uses two variables (see, e.g., [18]):

- The average response 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.

¹⁷ Each endpoint may be provisioned with a separate Call Agent address and port.

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

- It should double the estimated value of the average delay, AAD,
- It should compute a random value, uniformly distributed between 0.5 AAD and AAD,
- It should set the retransmission timer (RTO) to the minimum of:
 - the sum of that random value and N times the average deviation.
 - RTO_{max} , where the default value for RTO_{max} is 4 seconds.

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 subject to the needs of real-time communication. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

The initial value used for the retransmission timer is 200 milliseconds by default and the maximum value for the retransmission timer is 4 seconds by default. These default values may be altered by the provisioning process.

4.6 Piggy-Backing

There are cases when a Call Agent will want to send several messages at the same time to one or more endpoints in a gateway and vice versa. When several messages have to be sent in the same UDP packets, they are separated by a line of text that contain a single dot, as in for example:

```
200 2005 OK
.
DLCX 1244 aaln/2@rgw.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The piggy-backed messages **MUST** be processed as if they had been received in separate datagrams, however if a message (command or response) needs to be retransmitted, the entire datagram **MUST** be retransmitted, not just the missing message. The individual messages in the datagram **MUST** be processed in order starting with the first message.

Errors encountered in a message that was piggybacked **MUST NOT** affect any of the other messages received in that packet – each message is processed on its own.

5 Security

If unauthorized entities could use the MGCP, they would be able to set up unauthorized calls or interfere with authorized calls. Security is not provided as an integral part of MGCP. Instead MGCP assumes the existence of a lower layer providing the actual security. Security requirements and solutions for NCS are provided in a separate document, which at time of writing is not yet available. The remainder of this section is purely informational in that it illustrates one way security can be provided for NCS by the use of the IP security architecture, a.k.a. IPSEC.

5.1 Using IPSEC with MGCP

In a security architecture based on IPSEC, we expect that MGCP messages will always be carried over secure Internet connections, as defined in the IP security architecture defined in *RFC 1825*; using either the IP authentication header defined in *RFC 1826*; or the IP-encapsulating security payload defined in *RFC 1827*. The complete MGCP protocol stack would thus include the following layers:

MGCP
UDP
IP security (authentication or encryption)
IP
Transmission media

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 forbidding 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 protect, necessarily, 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.

Again, it should be noted that security requirements for embedded clients are specified in a separate document, which was not available at time of writing.

6 Acknowledgements

This specification was developed and influenced by numerous individuals representing many different vendors and organizations. PacketCable™ hereby wishes to thank everybody who participated directly or indirectly in this effort. In particular, PacketCable™ wants to recognize the authors of the original SGCP, MGCP, and IPDC specifications for the work they have done. Furthermore, PacketCable™ wishes to thank the following individuals for their involvement and contributions to this specification:

Vendor Authors:

Flemming Andreasen/Bellcore, Brian Bailey/3Com, Jack Fijolek/3Com, Bill Foster/Cisco, Christian Huitema/Bellcore, Jiri Matousek, David Oran/Cisco, Jeff Orwick/NetSpeak, John Pickens/Com21, Kurt Steinbrenner/Motorola, and Michael Thomas/Cisco.

CableLabs Team Lead:

David Bukovinsky/CableLabs

7 References

- [1] Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema, and Scott Pickett, *Media Gateway Control Protocol 0.1*, draft-huitema-mgcp-v0r1-03.txt, February 1, 1999.
- [2] Mauricio Arango, Christian Huitema, *Simple Gateway Control Protocol 1.1*, draft-huitema-MGCP-v1-02.txt, July 30, 1998.
- [3] P. Tom Taylor, Pat R. Calhoun, Allan C. Rubens, *IPDC Base Protocol*, draft-taylor-ipdc-00.txt, July 1998.
- [4] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 1889, January 1996.
- [5] Schulzrinne, H., *RTP Profile for Audio and Video Conferences with Minimal Control*, RFC 1890, January 1996
- [6] Handley, M, Jacobson, V., *SDP: Session Description Protocol*, RFC 2327, April 1998.
- [7] Handley, M., *SAP - Session Announcement Protocol*, Work in Progress.
- [8] Handley, M., Schooler, E., and H. Schulzrinne, *Session Initiation Protocol (SIP)*, Work in Progress.
- [9] Schulzrinne, H., Rao, A., and R. Lanphier, *Real-time Streaming Protocol (RTSP)*, RFC 2326, April 1998.
- [10] ITU-T, Recommendation Q.761, *Functional Description of the ISDN User Part of Signalling System No. 7*, (Malaga-Torremolinos, 1984; modified at Helsinki, 1993).
- [11] ITU-T, Recommendation Q.762, *General Function of messages and Signals of the ISDN User part of Signalling System No. 7*, (Malaga-Torremolinos, 1984; modified at Helsinki, 1993)
- [12] ITU-T, Recommendation H.323, *Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-guaranteed Quality-of-Service*.
- [13] ITU-T, Recommendation H.225, *Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems*.
- [14] ITU-T, Recommendation H.245, *Line Transmission of Non-telephone Signals*.
- [15] Atkinson, R., *Security Architecture for the Internet Protocol*, RFC 1825, August 1995.
- [16] Atkinson, R., *IP Authentication Header*, RFC 1826, August 1995.

- [17] Atkinson, R., *IP Encapsulating Security Payload (ESP)*, RFC 1827, August 1995.
- [18] Stevens, W. Richard, *TCP/IP Illustrated, Volume 1, The Protocols*, Addison-Wesley, 1994.
- [19] *RTP Parameters*, <http://www.isi.edu/in-notes/iana/assignments/rtp-parameters>
- [20] Bellcore, *Bellcore Notes on the Networks*, SR-2275.

Appendix A - Event Packages

This section defines an initial set of event packages for the various types of endpoints currently defined by PacketCable™ for embedded clients. Each package defines a package name for the package and event codes and definitions for each of the events in the package.

A.1 - Analog Access Lines

The following packages are currently defined for Analog Access Line endpoints:

- Line
- ADSI

A.1.1 Line Package

Package name: L

The following codes are used to identify events and signals for the “line” package for “analog access lines”:

Code	Description	Event	Signal	Additional Info
0-9,*,#,A, B,C,D	DTMF tones	√	BR	
aw	Answer tone	√	-	
bz	Busy tone	-	TO	Time-out = 30 seconds
cf	Confirmation tone	-	BR	
ci(ti, nu, na)	Caller Id	-	BR	“ti” denotes time, “nu” denotes number, and “na” denotes name
dl	Dial tone	-	TO	Time-out = 16 seconds
ft	Fax tone	√	-	
hd	Off-hook transition	P, S	-	
hf	Flash hook	P	-	
hu	On-hook transition	P, S	-	
L	DTMF long duration	√	-	
ld	Long duration connection	√	-	
mt	Modem tones	√	-	
mwi	Message waiting indicator	-	TO	Time-out = 16 seconds
oc	Operation complete	√	-	
ot	Off-hook warning tone	-	TO	Time-out = infinite
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive ringing (0..7)	-	TO	Time-out = 180 seconds
rbk(<xxx>)	Ringback on connection		TO	Time-out = 180 seconds “<xxx>” denotes a connection the ringback is to be played on
rg	Ringing	-	TO	Time-out = 180 seconds
ro	Reorder tone	-	TO	Time-out = 30 seconds
rs	Ringsplash	-	BR	
rt, rt(<xxx>)	Ring back tone	-	TO	Time-out = 180 seconds <xxx> denotes a connection the tone generation is to be synchronized with
sl	Stutter dial tone	-	TO	Time-out = 16 seconds
t	Timer	√	-	
TDD	Telecomm Devices for the Deaf (TDD) tones	√	-	
vmwi	Visual message waiting indicator	-	OO	
wt1, wt2, wt3, wt4	Call waiting tones	-	BR	
X	DTMF tones wildcard	√	-	Matches any of the digits “0-9”

For the events above, a P indicates that the event is persistent, and an S indicates that the event is an event-state that may be audited.

For the signals above, OO indicates the signal is of type On-Off, TO indicates the signal is of type Time-Out, and BR indicates the signal the signal is of type Brief.

Time-out values supplied are default time-out values. A value of zero indicates that the time-out period is infinite. The provisioning process may alter these default values.

Unless otherwise stated, all of the signals are applied to endpoints and audio generated by them is not forwarded on any connection the endpoint may have. ""

The definition of the individual events and signals are as follows:

DTMF tones (0-9,*,#,A, B,C,D): Detection and generation of DTMF signals is described in GR-506-CORE – LSSGR: SIGNALING, Section 15. It is considered an error to try and play DTMF tones on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Answer tone (aw): Answer tone is a tone that may be provided by a modem or fax that answers an incoming call. The tone consists of a sinewave signal at 2100 Hz - see ITU-T recommendation V.8.

Busy tone (bz): Station Busy is a combination of two AC tones with frequencies of 480 and 620 Hertz and levels of –24 dBm each, to give a combined level of –21 dBm. The cadence for Station Busy Tone is 0.5 seconds on followed by 0.5 seconds off, repeating. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.6. It is considered an error to try and play busy tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Confirmation tone (cf): Confirmation Tone uses the same frequencies and levels as dial tone (350 and 440 Hertz) but with a cadence of 0.1 second on, 0.1 second off repeated three times. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.4. It is considered an error to try and play confirmation tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Caller Id (ci(time, number, name)): See TR-NWT-001188, GR-30-CORE, and TR-NWT-000031. Each of the three fields are optional, however each of the commas will always be included.

- The **time** parameter is coded as “MM/DD/HH/MM”, where MM is a two-digit value for Month between 01 and 12, DD is a two-digit value for Day between 1 and 31, and Hour and Minute are two-digit values coded according to military local time, e.g., 00 is midnight, 01 is 1 a.m., and 13 is 1 p.m.
- The **number** parameter identifies the calling line number.

- The **name** parameter identifies the calling line name.

A “P” in the number or name field is used to indicate a private number or name, and an “O” is used to indicate an unavailable number or name.

Dial-tone (dl): Dial Tone is a combination of two continuous AC tones with frequencies of 350 and 440 Hertz and levels of –13dBm each to give a combined level of –10 dBm. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.1. It is considered an error to try and play dial-tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Fax tone (ft): The fax tone event is generated whenever a fax call is detected – see e.g., ITU-T recommendation T.30, or V.21.

Off-hook transition (hd): See GR-506-CORE – LSSGR: SIGNALING, Section 12.

Flash hook (hf): See GR-506-CORE – LSSGR: SIGNALING, Section 12.

On-hook transition (hu): See GR-506-CORE – LSSGR: SIGNALING, Section 12. The timing for the on-hook signal is for flash response enabled.

DTMF Long duration (L): The “DTMF Long duration” is observed when a DTMF signal is produced for a duration longer than two seconds. In this case, the gateway will detect two successive events: first, when the signal has been recognized, the DTMF signal, and then, 2 seconds later, the long duration signal.

Long duration connection (ld): The “long duration connection” is detected when a connection has been established for more than a certain period of time. The default value is 1 hour, however this may be changed by the provisioning process.

Modem tones (mt): The modem tone event is generated whenever a modem call is detected – see e.g., ITU-T recommendation V.8.

Message Waiting Indicator (mwi): Message Waiting indicator tone uses the same frequencies and levels as dial tone (350 and 440 Hertz at –13dBm each) but with a cadence of 0.1 second on, 0.1 second off repeated 10 times followed by steady application of dial tone. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.3. It is considered an error to try and play message waiting indicator on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

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, 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 may carry as a parameter the name of the signal that came to the end of its live time, as in:

O: L/oc(L/dl)

When the event is requested, the signal parameter can not be specified. When the package name is omitted, the default package name is assumed.

Off-hook warning tone (ot): Receiver Off Hook Tone (ROH Tone) is the irritating noise a telephone makes when it is not hung up correctly. ROH Tone is generated by combining four tones at frequencies of 1400 Hertz, 2060 Hertz, 2450 Hertz and 2600 Hertz at a cadence of 0.1 second on, 0.1 second off, repeating. GR-506-CORE, Section 17.2.8 contains details about required power levels. It is considered an error to try and play off-hook warning tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Distinctive ringing (r0, r1, r2, r3, r4, r5, r6 or r7): See GR-506-CORE – LSSGR: SIGNALING, Section 14. Default values for r1 to r5 are as defined for distinctive ringing pattern 1 to 5 as defined in GR-506-CORE. The provisioning process may define the ringing cadence for each of these signals. It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

Ringback on connection (rbk(<xxx>)): Ringback on connection tone is generated by the called party and is inserted into the outgoing media stream for the connection identified by <xxx> only, *regardless* of the mode of the connection¹⁸. The wildcard “\$” can be used to denote “the current connection”¹⁹. This notation **MUST NOT** be used outside a connection handling command – the wildcard refers to the connection in question for the command. The definition of the tone is defined by the national characteristics of the Audible Ring Tone, and **MAY** be established via provisioning. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.5.

Ringng (rg): See GR-506-CORE – LSSGR: SIGNALING, Section 14. The provisioning process may define the ringing cadence. It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

¹⁸ The use of this signal will consume bandwidth for ringback tones. However, it guarantees perfect synchronization with the media generated by the person answering the phone without requiring that the calling party does not receive any RTP packets whatsoever prior to the called party answering.

¹⁹ This can be used to refer to a connection that is being created at the same time the event is requested, e.g. by a CreateConnection command containing a notification request.

Reorder tone (ro): Reorder tone is a combination of two AC tones with frequencies of 480 and 620 Hertz and levels of –24 dBm each, to give a combined level of –21 dBm. The cadence for reorder tone is 0.25 seconds on followed by 0.25 seconds off, repeating continuously. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.7. It is considered an error to try and play reorder tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Ringsplash (rs): Ringsplash, also known as “Reminder ring” is a burst of ringing that may be applied to the physical forwarding line (when idle) to indicate that a call has been forwarded and to remind the user that a Call Forwarding subfeature is active. In the US, it is defined to be a 0.5(-0,+0.1) second burst of power ringing. See TR-TSY-000586 – Call Forwarding Subfeatures. It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

Ring back tone (rt, rt(<xxx>)): Audible Ring Tone is a combination of two AC tones with frequencies of 440 and 480 Hertz and levels of –19 dBm each, to give a combined level of –16 dBm. In the U.S. the cadence for Audible Ring Tone is defined to be 2 seconds on followed by 4 seconds off. The definition of the tone is defined by the national characteristics of the Ringback Tone, and MAY be established via provisioning. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.5. It is considered an error to try and play ring back tones on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

The “rt(<xxx>)” allows the generation of ring back to be synchronized with the reception of media on the connection referenced by “<xxx>”, where “\$” can be used to denote “the current connection”²⁰. This notation MUST NOT be used outside a connection handling command – the wildcard refers to the connection in question for the command. As soon as any incoming RTP media packet is detected on the connection referenced, the ring back tone will stop²¹.

Stutter Dial tone (sl): Stutter Dial Tone (also called Recall Dial Tone) is generated by supplying Confirmation Tone, followed by continuous Dial Tone. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.2. It is considered an error to

²⁰ This can be used to refer to a connection that is being created at the same time the event is requested, e.g. by a CreateConnection command containing a notification request.

²¹ This allows bandwidth to not be used for ringback tones, while maintaining perfect synchronization with the media generated by the person answering the phone. It does however also require that no RTP packets whatsoever are received prior to the called party answering.

try and play stutter dial tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Timer (t): As described in Section 3.1.5, timer T is a provisionable timer that can only be cancelled by DTMF input. When timer T is used with the “accumulate according to digit map” action, the timer is not started until the first digit is entered, and the timer is restarted after each new digit is entered until either a digit map match or mismatch occurs. In this case, timer T functions as an inter-digit timer and takes on one of two values, T_{par} or T_{crit} . When at least one more digit is required for the digit string to match any of the patterns in the digit map, timer T takes on the value T_{par} , corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value T_{crit} corresponding to critical timing. An example use is:

```
S:    dl
R:    [0-9T](D)
```

When timer T is used without the “accumulate according to digit map” action, timer T takes on the value T_{crit} , and the timer is started immediately and simply cancelled (but not restarted) as soon as a digit is entered. In this case, timer T can be used as an inter-digit timer when overlap sending is used, e.g.:

```
R:    [0-9](N), T(N)
```

Timer T can be specified both with and without the “accumulate according to digit map” action in a single request, as in:

```
R:    [0-9T](D), T(N)
```

In that case, timer T will be started immediately. If a digit is entered before timer T expires, timer T will be cancelled, however since timer T is also specified with the digit map action, timer T will subsequently be restarted.

The default value for T_{par} is 16 seconds and the default value for T_{crit} is 4 seconds. The provisioning process may alter both of these.

Telecomm Devices for the Deaf tones (TDD): The TDD event is generated whenever a TDD call is detected – see e.g., ITU-T recommendation V.18.

Visual Message Waiting Indicator (vmwi): The transmission of the VMWI messages will conform to the requirements in TR-H-000030 - “Section 2.3.2 - “On-hook Data Transmission Not Associated with Ringing”, and the CPE guidelines in SR-TSV-002476. VMWI messages will only be sent from the embedded client to the attached equipment when the line is idle. If new messages arrive while the line is busy, the VMWI indicator message will be delayed until the line goes back to the idle state. The Call Agent should periodically refresh the CPE’s visual indicator. See TR-NWT-001401 – Visual Message Waiting Indicator Generic Requirements; and GR-30-CORE - Voiceband Data Transmission Interface.

Call Waiting tone1 (wt1, .., wt4): Call Waiting tones are defined in GR-506-CORE, Section 14.2 – the number refers to the tone pattern used. The default Call Waiting tone is a 440-Hz tone applied for 300 ± 50 ms. The talking path should be interrupted for a maximum of 400 ms for the application of CW tone. The Call Agent implements the actual call waiting service - see TR-TSY-000571 – Call Waiting. It is considered an error to try and apply call waiting tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

DTMF tones wildcard (X):

The DTMF tones wildcard matches any DTMF digit between 0 and 9.

A.1.2 ADSI Package

Package name: S

Code	Description	Event	Signal	Additional Info
adsi(string)	ADSI display	-	BR	

The definition of the events and signals are as follows:

ADSI display (adsi(string)):

See GR-1273-CORE - ANALOG DISPLAY SERVICES INTERFACE (ADSI) SPCS/SERVER GENERIC REQUIREMENTS (A MODULE OF ADSI, FR-12)

A.2 Video

Event packages for video will be provided in a future version of this document.

Appendix B - Mode Interactions

An MGCP connection 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 should 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”

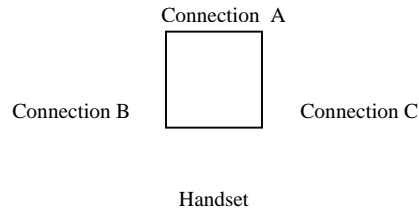
Connections in “network loopback”, “network continuity test”, or “inactive” modes are not affected by connections in the “active” modes. The Table uses the following conventions:

- A_{in} is the incoming media stream from Connection A
- B_{in} is the incoming media stream from Connection B
- H_{in} is the incoming media stream from the Handset Microphone
- A_{out} is the outgoing media stream to Connection A
- B_{out} is the outgoing media stream to Connection B
- H_{out} is the outgoing media stream to the Handset earpiece
- NA indicates No Stream whatever

		Connection A Mode					
		sendonly	recvonly	sendrecv	confrnce	inactive	netwloop/ netwtest
Connection B Mode	sendonly	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$
	recvonly		$A_{out}=NA$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=NA$ $H_{out}=B_{in}$
	sendrecv			$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$
	confrnce				$A_{out}=H_{in}+B_{in}$ $B_{out}=H_{in}+A_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$
	inactive					$A_{out}=NA$ $B_{out}=NA$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}=NA$ $H_{out}=NA$
	netwloop/ netwtest						$A_{out}=A_{in}$ $B_{out}=B_{in}$ $H_{out}=NA$

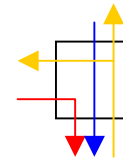
If there are three or more “active” channels they will still interact as defined in the table above with the outgoing media streams mixed for each interaction. (Union of all streams)
 If internal resources are used up and the streams cannot be mixed, the gateway should return a resources Not available error.

These connections can be graphically represented as such:

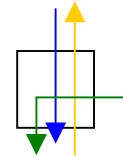


For example, if Connection A is Sendrecv, Connection B is confrnce, and Connection C is recvnly, from the above table the outputs in each mode will be:

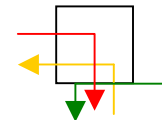
A to B Interaction: $B_{out} = H_{in}$ $A_{out} = H_{in}$ $H_{out} = A_{in} + B_{in}$



A to C Interaction: $A_{out} = H_{in}$ $C_{out} = NA$ $H_{out} = A_{in} + C_{in}$

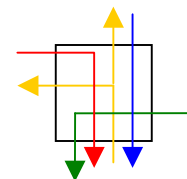


B to C Interaction: $B_{out} = H_{in}$ $C_{out} = NA$ $H_{out} = B_{in} + C_{in}$



Taking the Union of all streams in each output we get:

- $A_{out} = H_{in}$
- $B_{out} = H_{in}$
- $C_{out} = NA$
- $H_{out} = B_{in} + A_{in} + C_{in}$



For clarity, the table described above is repeated below in graphical form:

		Connection A Mode (top)					
		sendonly	recvonly	sendrecv	confnce	inactive	netwloop/ netwtest
Connection B Mode (Left)	sendonly						
	recvonly						
	sendrecv						
	confnce						
	inactive						
	netwloop/ netwtest						

Appendix C - 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.

C.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 0.1 NCS
1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hd(N)
S: 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 empty²², 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” and “lockstep” state will include fax tones in addition to the “requested events” and persistent events:

```
RQNT 1202 aaln/1@rgw-2567.whatever.net MGCP 0.1 NCS
1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hd(A, E(S(dl), R(oc, hu, [0-9#*T](D))))
D: (0T|00T|[1-
xxx|8xxxxxxx|#xxxxxxx|*xx|91xxxxxxxxxxx|9011x.T)
S:
Q: process
T: ft
```

The response indicates that the transaction was successful:

```
200 1202 OK
```

²² It could have been omitted as well.

C.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 0.1 NCS
1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: hd,9,1,2,0,1,8,2,9,4,2,6,6
```

The Notify response indicates that the transaction was successful:

```
200 2002 OK
```

C.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 u-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 0.1 NCS
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
c=IN IP4 128.96.41.1
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 0.1 NCS
1.0
```

```

C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789AD
R: hd
S: rg

v=0
c=IN IP4 128.96.41.1
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 2005 Phone off-hook ""
```

C.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 0.1 NCS
1.0
C: A3C47F21456789F0
I: FDE234C8
N: ca@cal.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 until it detects audio on the connection specified for the ring-back signal:

```

MDCX 1210 aaln/1@rgw-2567.whatever.net MGCP 0.1 NCS
1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: hu
S: rt(FDE234C8)

v=0

```

```
c=IN IP4 128.96.63.25
m=audio 1296 RTP/AVP 0
```

The response indicates that the transaction was successful: ""

```
200 1206 OK
```

C.5 – DeleteConnection (From the Call Agent)

In this example, the Call Agent simply instructs the embedded client to delete the connection FDE234C8 on the endpoint specified:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 0.1 NCS
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
```

C.6 – DeleteConnection (From the Embedded Client)

In this example, the embedded client 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 0.1 NCS
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
```

C.7 – DeleteConnection (Multiple Connections From the Call Agent)

In the first example, the Call Agent instructs the embedded client to delete all connections related to call "A3C47F21456789F0" on the specified endpoint:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 0.1 NCS
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 embedded client to delete all connections related to all of the endpoints specified:

```
DLCX 1210 aaln/*@rgw-2567.whatever.net MGCP 0.1 NCS
1.0
```

The response indicates success:

```
250 1210 OK
```

C.8 – AuditEndpoint

In the first example, the Call Agent wants to learn what endpoints are present on the embedded client 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 0.1 NCS 1.0
```

The embedded client 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 0.1 NCS
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;confrnce;ne
twloop;          netwtest
A: a:G723, p:30-90, e:on, s:on, v:L;S,
        m:
sendonly;recvonly;sendrecv;inactive;confrnce;netwloop

```

In the third example, the Call Agent audits all possible information for the endpoint:

```

AUEP 2002 aaln/1@rgw-2567.whatever.net MGCP 0.1 NCS
1.0
F: R,D,S,X,N,I,T,O,ES

```

The response indicates success:

```

200 2002 OK
R: L/hu,oc(N),[0-9](N)
D:
S: vmwi(+)
X: 0123456789B1
N: [128.96.41.12]
I: 32F345E2
T: ft
O: hd,9,1,2
ES: 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 “vmvi” 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 “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.

C.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 0.1 NCS
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@cal.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
c=IN IP4 128.96.63.25
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 0.1 NCS
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
c=IN                                IP4                                128.96.63.25
m=audio                             1296                             RTP/AVP                                0

v=0

```

C.10 – RestartInProgress

The first example illustrates a RestartInProgress message sent by an embedded client 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 0.1 NCS
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 embedded client informs the Call Agent, that all of the embedded client's endpoints are being placed in service in 0 seconds, i.e., they are back in service. The delay could have been omitted as well:

```
RSIP 1204 *@rgw-2567.whatever.net MGCP 0.1 NCS 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" (see Section 3.4.3.4), this time going to Call Agent "CA-1@whatever.net".

Appendix D – Example Call Flow

In this section we provide an example call flow between two embedded clients, EC-1 and EC-2. It should be noted, that this call flow, although a valid one, is merely an example that may or may not be used in practice.

In the call flow below, CA refers to the Call Agent, CDB refers to a configuration database, and ACC refers to an accounting database.

Usr-1	EC-1	CA	CDB	ACC	EC-2	Usr-2
	<-	Notification Request				
	Ack	->				
Off-hook	Notify	->				
	<-	Ack				
(Dial-tone)	<-	Create Connection + Notification Request				
	Ack(SDP1)	->				
Digits	Notify	->				
	<-	Ack				
(progress)	<-	Notification Request				
	Ack	->				
		Query(E.164)	->			
		<-	IP			
		Create Connection(SDP1) + Notification Request	---	---	->	(ringing)
		<-	---	---	Ack(SDP2)	
	<-	Modify Connection(SDP2) + Notification Request				
	Ack	->				
		<-	---	---	Notify	Off-hook
		Ack	---	---	-->	
	<-	ModifyConnection + Notification Request				
	Ack	->				
	(cut in)	Call start	---	-->		
		Notification Request	---	---	->	
		<-	---	---	Ack	
		(Call Established)				
		<-	---	---	Notify	on hook
		Ack	---	---	->	
	<-	Delete Connection				
		Delete Connection	---	---	->	
	Ack (Perf)	->				

Usr-1	EC-1	CA	CDB	ACC	EC-2	Usr-2
	Data)					
		←	---	---	Ack(Perf data)	
		Call end	---	→		
		Notification Request	---	---	→	
		←	---	---	Ack	
On-hook	Notify	→				
	←	Ack				
	←	Notification Request				
	Ack	→				

During these exchanges the NCS profile of MGCP is used by the Call Agent to control both embedded clients. The exchanges occur on two sides.

The first command is a NotificationRequest, sent by the Call Agent to the ingress embedded client. The request will consist of the following lines:

```
RQNT 1201 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
N: ca@cal.whatever.net:5678
X: 0123456789AB
R: hd
```

The embedded client, at that point, is instructed to look for an off-hook event, and to report it. It will first send a response to the command, repeating in the response the transaction id that the Call Agent attached to the query and providing a return code indicating success:

```
200 1201 OK
```

When the off hook event is noticed, the embedded client sends a Notify message to the Call Agent:

```
NTFY 2001 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
N: ca@cal.whatever.net:5678
X: 0123456789AB
O: hd
```

The Call Agent immediately acknowledges the notification:

```
200 2001 OK
```

The Call Agent examines the services associated to an off hook event for this endpoint (it could take special actions in the case of a direct line, no current subscription, etc.). In most cases, it will send a combined CreateConnection and NotificationRequest command to create a connection, provide dial-tone, and collect DTMF digits²³:

```
CRCX 1202 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hu, [0-9#*T](D)
D: (0T | 00T | [2-9]xxxxxx | 1[2-9]xxxxxxxxxx |
011xx.T)
S: dl
```

The embedded client acknowledges the transaction, sending back the identification of the newly created connection and the session description used to receive audio data:

```
200 1202 OK
I: FDE234C8

v=0
c=IN IP4 128.96.41.1
m=audio 3456 RTP/AVP 0
```

The SDP specification, in our example, specifies the address at which the embedded client is ready to receive audio data (128.96.41.1), the transport protocol (RTP), the RTP port (3456) and the audio profile (AVP). The audio profile refers to RFC 1890, which defines that the payload type 0 has been assigned for G.711 u-law transmission (also, see [19]).

The embedded client will start accumulating digits according to the digit map. When a digit map match subsequently occurs, the embedded client will notify the observed events to the Call Agent:

```
NTFY 2002 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: 1,2,0,1,8,2,9,4,2,6,6
```

²³ The actual digit map depends on dialing plan in the local area as well as services subscribed to. The digit map presented should be considered an example digit map only.

The Call Agent immediately acknowledges that notification.

```
200 2002 OK
```

At this stage, the Call Agent will send a NotificationRequest, to stop collecting digits yet continue to watch for an on-hook transition:

```
RQNT 1203 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
X: 0123456789AD
R: hu
```

The embedded client immediately acknowledges that command.

```
200 1203 OK
```

The Call Agent must now create a connection on the egress embedded client, EC-2, and ring the phone attached to the embedded client as well. It does so by sending a combined CreateConnection and NotificationRequest command to the embedded client:

```
CRCX 2001 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789B0
R: hd
S: rg

v=0
c=IN IP4 128.96.41.1
m=audio 3456 RTP/AVP 0
```

The egress embedded client, at that point, is instructed to ring the phone, and to look for an off-hook event, and report it. The off-hook event and ringing signal are synchronized, so when the off-hook event occurs, ringing will stop. The create connection portion of the command has the same parameters as the command sent to the ingress embedded client, with two differences:

- The endpoint identifier points towards the outgoing circuit.
- The message carries the session description returned by the ingress embedded client,

- Because the session description is present, the “mode” parameter is set to “send/receive”.

We observe that the call identifier is identical for the two connections. This is normal since the two connections belong to the same call.

The egress embedded client will acknowledge the command, sending in the session description its own parameters such as address, ports and RTP profile as well as the connection identifier for the new connection:

```
200 2001 OK
I: 32F345E2

v=0
c=IN IP4 128.96.63.25
m=audio 1297 RTP/AVP 0
```

The Call Agent will relay the information to the ingress embedded client, and instruct it to generate local ringback tones, using a combined ModifyConnection and NotificationRequest command:

```
MDCX 1204 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: hu
S: rt(FDE234C8)

v=0
c=IN IP4 128.96.63.25
m=audio 1297 RTP/AVP 0
```

The embedded client immediately acknowledges the modification:

```
200 1204 OK
```

At this stage, the Call Agent has established a half duplex transmission path. The phone attached to the ingress embedded client will be able to receive the signals, such as tones or announcements, that may be generated in case of any errors, as well as the initial speech that most likely will be generated when the egress user answers the phone.

When the off hook event is observed, the egress embedded client sends a Notify message to the Call Agent:

```
NTFY 3001 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
X: 0123456789B0
O: hd
```

The call agent immediately acknowledges that notification.

```
200 3001 OK
```

The Call agent now sends a combined ModifyConnection and NotificationRequest to the ingress embedded client, to place the connection in send/receive mode and stop the ringback tones:

```
MDCX 1206 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
M: sendrecv
X: 0123456789AF
R: hu
```

The embedded client immediately responds to the command:

```
200 1206 OK
```

In parallel, the Call Agent asks the egress embedded client to notify the occurrence of an on-hook event. It does so by sending a NotificationRequest to the embedded client²⁴:

```
RQNT 2002 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
X: 0123456789B1
R: hu
```

The embedded client immediately responds to the command:

```
200 2002 OK
```

At this point, the call is fully established.

²⁴ It should be noted, that although on-hook is a persistent event, lockstep mode requires the Call Agent to send a new NotificationRequest to the embedded client.

At some later point in time, the phone attached to the egress embedded client, in our scenario, goes on-hook. This event is notified to the Call Agent, according to the policy received in the last NotificationRequest by sending a Notify command:

```
NTFY 2003 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
X: 0123456789B1
O: hu
```

The Call Agent immediately responds to the command:

```
200 2003 OK
```

The Call Agent now determines that the call is ending, and it therefore sends both embedded clients a DeleteConnection command:

```
DLCX 1207 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

```
DLCX 2004 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
C: A3C47F21456789F0
I: 32F345E2
```

The embedded clients will respond with acknowledgements that include the connection parameters for the connection:

```
250 1207 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27,
LA=48
```

```
250 2004 OK
P: PS=790, OS=45700, PR=1230, OR=61875, PL=15, JI=27,
LA=48
```

The Call Agent will also issue a new NotificationRequest to the egress embedded client, to be ready to receive the next off-hook event detected by the embedded client:

```
RQNT 2005 aaln/1@ec-2.whatever.net MGCP 0.1 NCS 1.0
X: 0123456789B2
R: hd
```

The embedded client will acknowledge this message:

```
200 2005 OK
```

Finally, the ingress embedded client hangs up the phone thereby generating a Notify message to the Call Agent:

```
NTFY 1208 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0  
X: 0123456789AF  
O: hu
```

The Call Agent immediately responds to the command:

```
200 1208 OK
```

The Call Agent will then issue a new NotificationRequest to the ingress embedded client, to be ready to receive the next off-hook event detected by the embedded client:

```
RQNT 1209 aaln/1@ec-1.whatever.net MGCP 0.1 NCS 1.0  
X: 0123456789B3  
R: hd
```

The embedded client will acknowledge this message:

```
200 1209 OK
```

Both embedded clients, at this point, are ready for the next call.

Appendix E - Known Issues

This PacketCable specification is part of a larger collection of specifications being issued throughout 1999 to provide advanced, real-time, multimedia services over a cable infrastructure. Currently the following specifications are targeted for 2-3Q1999 release:

- ◆ Quality of service [HFC Access network signaling, backbone QoS]
- ◆ Embedded Device Provisioning
- ◆ Security Protocols and Mechanisms
- ◆ Codec Specification
- ◆ Network Management
- ◆ Billing data collection
- ◆ PSTN Gateway Interfaces

Additional specifications are also under consideration for domain interconnection, address/directory services, additional backoffice services, etc.

This specification represents the initial call model supported by a PacketCable network. The architecture currently under development will address the addition of a complimentary client-based call signaling model, where call state will be centralized in the client devices. Both call models are being supported on the core PacketCable infrastructure to support the varying needs and desires of our member companies for services and network devices.

Based on the upcoming release of the core PacketCable infrastructure specifications, Engineering Change Orders [ECOs] are expected to be submitted according to formal CableLabs procedures, and cause **minor** modifications in this specification, for the following areas [contents within brackets represent likely areas of focus affecting this specification, and are subject to change]:

- ◆ Quality of Service [two stage, pre-provisioned, and dynamic QoS models]
- ◆ Security [signaling and media security]
- ◆ Codecs [mandatory codecs to be supported on a PacketCable network]
- ◆ Provisioning

Due to the evolving nature of this technology, additional ECOs may result in minor changes based on future specification activities and interoperability initiatives.

