INTERNATIONAL TELECOMMUNICATION UNION

ITU-T J.162

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(03/2001)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

IPCablecom

Network call signalling protocol for the delivery of time-critical services over cable television networks using cable modems

ITU-T Recommendation J.162

ITU-T J-SERIES RECOMMENDATIONS

CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

General Recommendations	J.1–J.9
General specifications for analogue sound-programme transmission	J.10–J.19
Performance characteristics of analogue sound-programme circuits	J.20–J.29
Equipment and lines used for analogue sound-programme circuits	J.30–J.39
Digital encoders for analogue sound-programme signals	J.40–J.49
Digital transmission of sound-programme signals	J.50–J.59
Circuits for analogue television transmission	J.60–J.69
Analogue television transmission over metallic lines and interconnection with radio-relay links	J.70–J.79
Digital transmission of television signals	J.80–J.89
Ancillary digital services for television transmission	J.90–J.99
Operational requirements and methods for television transmission	J.100–J.109
Interactive systems for digital television distribution	J.110–J.129
Transport of MPEG-2 signals on packetised networks	J.130–J.139
Measurement of the quality of service	J.140–J.149
Digital television distribution through local subscriber networks	J.150–J.159
IPCablecom	**J.160–J.179**
Miscellaneous	J.180–J.199
Application for Interactive Digital Television	J.200–J.209

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

ITU-T Recommendation J.162

Network call signalling protocol for the delivery of time-critical services over cable television networks using cable modems

Summary

Many cable television operators are upgrading their facilities to provide two-way capability and using this capability to provide high-speed IP data services per ITU-T J.83 and ITU-T J.112. These operators now want to expand the capability of this delivery platform to include a variety of time-critical services. This Recommendation is one of a series of Recommendations required to achieve this goal. It provides a network-based call signalling protocol necessary to establish connections.

Source

ITU-T Recommendation J.162 was prepared by ITU-T Study Group 9 (2001-2004) and approved under the WTSA Resolution 1 procedure on 9 March 2001.

FOREWORD

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

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

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

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

NOTE

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

INTELLECTUAL PROPERTY RIGHTS

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

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

CONTENTS

ITU-T Recommendation J.162

Network call signalling protocol for the delivery of time-critical services over cable television networks using cable modems

1 Scope

This Recommendation describes a profile of an application programming interface, Media Gateway Controller Interface (MGCI), and a corresponding protocol, Media Gateway Control Protocol (MGCP), for controlling voice-over-IP (VoIP) embedded clients from external call control elements. The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and is handled by external call control elements. The profile, as described in this Recommendation, will be referred to as the Network-based Call Signalling (NCS) Protocol.

This Recommendation is based on the Media Gateway Control Protocol (MGCP) 1.0 RFC 2705, which is the result of a merge of the Simple Gateway Control Protocol, and the IP Device Control (IPDC) family of protocols, as well as input generated by the people that developed this profile.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revisions; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

2.1 Normative references

- ITU-T J.83 (1997), *Digital multi-programme systems for television, sound and data services for cable distribution*.
- ITU-T J.112 Annex A (2001), *Digital video broadcasting: DVB interaction channel for Cable TV (CATV) distribution systems*.
- ITU-T J.112 Annex B (2001), *Data-over-cable service interface specifications: Radio-frequency interface specification*.
- ITU-T J.160 (Draft), *Architectural framework for the delivery of time-critical services over cable television networks using cable modems*.
- ITU-T J.161 (2001), *Audio codec requirements for the provision of bidirectional audio service over cable television networks using cable modems*.
- ITU-T J.163 (2001), *Dynamic quality of service for the provision of real-time services over cable television networks using cable modems*.
- IETF RFC 821 (1982), *Simple Mail Transfer Protocol*.
- IETF RFC 1034 (1987), *Domain names – Concepts and facilities*.
- IETF RFC 1889 (1996), *RTP: A Transport Protocol for Real-Time Applications*.
- IETF RFC 2045 (1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*.
- IETF RFC 2234 (1997), *Augmented BNF for Syntax Specifications: ABNF*.

– IETF RFC 2327 (1998), *SDP: Session Description Protocol*.

– IETF RFC 2543 (1999), *SIP: Session Initiation Protocol*.

NOTE – The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

2.2 Informative references

– IETF RFC 1890 (1996), *RTP Profile for Audio and Video Conferences with Minimal Control*.

– IETF RFC 2705 (1999), *Media Gateway Control Protocol (MGCP) Version 1.0*.

3 Terms and definitions

This Recommendation defines the following terms:

3.1 cable modem: A cable modem is a layer two termination device that terminates the customer end of the J.112 connection.

3.2 IPCablecom: An ITU-T project that includes an architecture and a series of Recommendations that enable the delivery of time-critical interactive services over cable television networks.

3.3 MUST: The term "MUST" or "MUST NOT" is used as a convention in the present Recommendation to denote an absolutely mandatory aspect of the specification.

4 Abbreviations

This Recommendation uses the following abbreviations:

API Application Programming Interface

CPE Customer Premise Equipment

DTMF Dual Tone Multi Frequency

IP Internet Protocol

MGCI Media Gateway Controller Interface

MGCP Media Gateway Control Protocol

MIB Management Information Base

MTA Media Terminal Adaptor

MWD Maximum Waiting Delay

NCS Network Call Signalling

PSTN Public Switched Telephone Network

RTP Real-time Protocol

SDP Session Description Protocol

UDP User Datagram Protocol

5 Introduction

This Recommendation 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 analogue access lines to a voice-over-IP (VoIP) network;
- one or more video lines to a VoIP network (for further study).

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. analogue 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 Recommendation does not define a mechanism for synchronizing Call Agents, although future IPCablecom 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 analogue POTS connection to a phone, key system, PBX, etc. A gateway that terminates residential POTS lines (to phones) is called a *residential gateway,* an *embedded client*, or an *MTA*. Embedded clients may optionally support video as well.

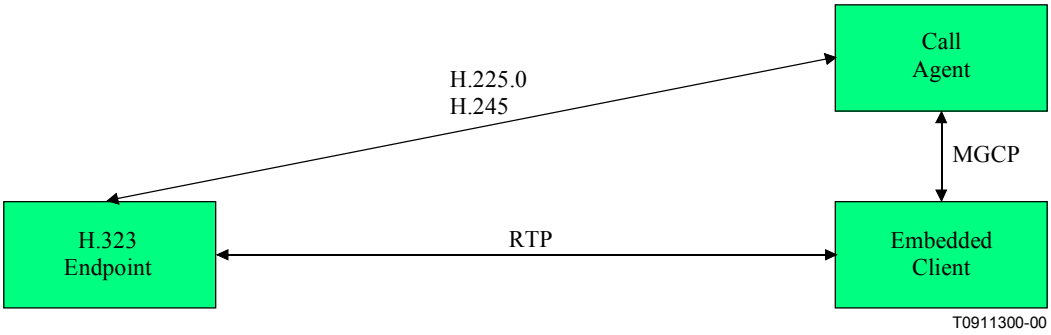
An example of a virtual endpoint is an audio source in an audio-content server. Creation of physical endpoints requires hardware installation, while creation of virtual endpoints can be accomplished by software. However, the NCS profile of MGCP only addresses physical endpoints.

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. The association is established by creating the connection as two halves: one on the origination endpoint, and one on the terminating endpoint.

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 know 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.

5.1 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 signalling and call processing functions. As a consequence, the Call Agent implements the "signalling" 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. The H.225.0 call signalling and H.245 media signalling is therefore routed to the Call Agent.

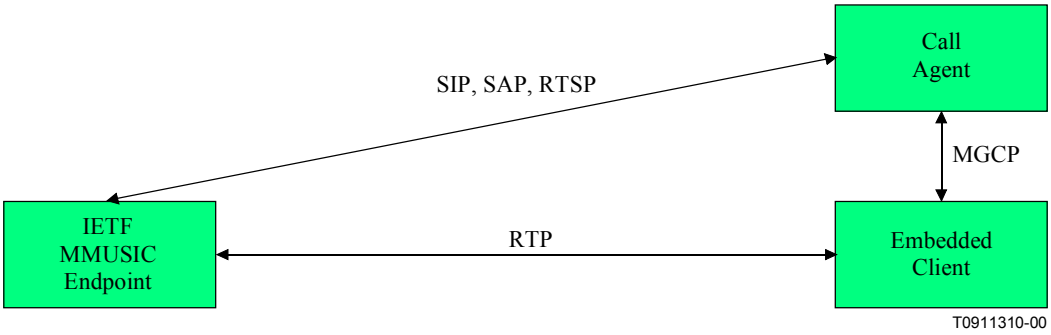
5.2 Relation with IETF standards

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

- the session description protocol (SDP), RFC 2327;
- the session announcement protocol (SAP), RFC 2974: work in progress;
- the session initiation protocol (SIP), RFC 2543;
- the real-time streaming protocol (RTSP), RFC 2326.

The latter three specifications are, in fact, alternative signalling 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 voice communication and embedded client users to access sessions set up using SAP, SIP, or RTSP defined by the IETF MMUSIC Working Group. The Call Agent provides for signalling 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.

6 Media Gateway Controller Interface (MGCI)

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

6.1 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.

6.1.1 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 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 analogue 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 ("*", "$"), and white space characters.
- Wildcarding is represented either by an asterisk ("*") or a dollar sign ("$") for the terms of the naming path which are to be wildcarded. Thus, if the full local endpoint name looks like:

  `term1/term2/term3`

  and one of the terms of the local endpoint name is wildcarded, then the local endpoint name looks like this:

  `term1/term2/*` if `term3` is wildcarded.

  `term1/*/*` if `term2` and `term3` are wildcarded.

  In each of the examples, a dollar sign could have appeared instead of the asterisk.
- Wildcarding is only allowed from the right; thus, if a term is wildcarded, then all terms to the right of that term must be wildcarded as well.
- In cases where mixed dollar sign and asterisk wildcards are used, dollar-signs are only allowed from the right; thus, if a term had a dollar sign wildcard, all terms to the right of that term must also contain dollar sign wildcards.
- 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.

6.1.1.1 Embedded client endpoint names

Endpoints in embedded clients MUST support the additional naming conventions specified in this clause.

Embedded clients MAY support one or more endpoint-types including the following:

- `Analogue Telephone` – The analogue telephone is represented as an analogue access line (aaln). This is basically the equivalent of an analogue telephone line as known in the PSTN.
- `Video` – The details of the video device-type are for further study.
- `Basic Access ISDN` – The details of the ISDN device-type are for further study.

6.1.1.1.1 Analogue access line endpoints

In addition to the naming conventions specified above, local endpoint names for endpoints of type "analogue access line" (`aaln`) for embedded clients must adhere to the following:

- Local endpoint names contain at least one and, at most, two terms.
- term1 MUST 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 MUST be a number from one to the number of analogue access lines supported by the embedded client in question. The number thus identifies a specific analogue 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 analogue access line local endpoint names could thus be:

– `aaln/1` The first analogue access line on the embedded client in question.

– `aaln/2` The second analogue access line on the embedded client in question.

– `aaln/$` Any analogue access line on the embedded client in question.

– `aaln/*` All analogue access lines 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.

6.1.2 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. Call identifiers with a maximum length of 32 MUST be supported.

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 Recommendation. 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.

6.1.3 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. Connection names with a maximum length of 32 characters MUST be supported.

6.1.4 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:

```
ca1@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" MUST 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 explicitly[1], 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".

Clause 6.4 contains a more detailed description of reliability and fail-over.

6.1.5 Digit maps

The Call Agent can ask the gateway to collect digits dialled by the user. This facility is intended to be used for analogue 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. Endpoints MUST support digit maps as defined in this clause.

An alternative procedure involves the gateway notifying the Call Agent of the dialled digits as soon as they are dialled, a.k.a. overlap sending. However, such a procedure generates a large number of interactions. It is preferable to accumulate the dialled 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 for the area in which the gateway resides. Thus the actual digit map used may differ between regions. 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)
```

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

The formal syntax of the digit map is described by the following BNF notation:

```
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 digit map, 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 string[2]. 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.

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 Agent[3] 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 map[4], 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.

---

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

3 The list of digits may include other events as well – see 6.4.3.1.

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

Digit maps can be provided to the gateway by the Call Agent, whenever the Call Agent instructs the gateway to listen for digits. Again, it should be noted that the details of the digit map used will depend on the area in which the gateway resides and thus the digit map is programmable. Digit maps, when provided by the Call Agent, MUST be as defined in this clause.

6.1.6 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 analogue 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. Package names and event codes are case insensitive strings of letters, digits, and hyphens, with the restriction that hyphens MUST NOT be the first or last character in a name. Some event codes may need to be parameterized with additional data, which is accomplished by adding the parameters between a set of parentheses. 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 analogue access line with the example line package (package name "X") being the default package, the following two event names are considered equal:

- X/dl dial-tone in the example line package for an analogue access line;
- dl dial-tone in the example line package (default) for an analogue access line.

Annex A defines an initial set of packages. Additional package names and event codes may be defined by and/or registered with IPCablecom. Any change to the packages defined in this Recommendation 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 6.3.1) and if they contain auditable event-states (e.g. off-hook, see 6.3.8.1). 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 6.3.1.

In addition to IPCablecom packages, implementers MAY gain experience by defining experimental packages. The package name of experimental packages MUST begin with the two characters "x-" or "X-"; IPCablecom MUST NOT register package names that start with these two characters. An embedded client that receives a command referring to an unsupported package MUST return an error (error code 518 – unsupported package).

Package names and event codes support one wildcard 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 refer to all events in the package in question. For example:

- X/all refers to all events in the example line package for an analogue access line;
- */all for an analogue 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.

Events and signals are by default detected and generated on endpoints; however, some events and signals may be detected and generated on connections in addition to or instead of on an endpoint. For example, endpoints may be asked to provide a ringback tone on a connection. In order for an event or signal to be able to be detected or generated on a connection, the definition of the event/signal MUST explicitly define that the event/signal can be detected or generated on a connection.

When a signal shall be applied on a connection, the name of the connection is added to the name of the event, using an "at" sign (@) as a delimiter, as in:

```
X/rt@0A3F58
```

The wildcard character "*" (asterisk) can be used to denote "all connections" on the affected endpoint(s). When this convention is used, the gateway MUST generate or detect the event on all the connections that are connected to the endpoint(s). An example of this convention is:

```
X/rt@*
```

The wildcard character "$" (dollar sign) can be used to denote "the current connection". This convention MUST NOT be used unless the event notification request is "encapsulated" within a CreateConnection or ModifyConnection command. When the convention is used, the gateway MUST generate or detect the event on the connection that is currently being created or modified. An example of this convention is:

```
X/rt@$
```

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

```
*/all@*
```

can be used to designate all events on all connections for the affected endpoint(s).

6.2 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 Recommendation, SDP descriptions MUST follow the conventions delineated in the session description protocol (SDP), which is now an IETF-proposed standard 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.

6.3 Gateway control functions

This clause 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 controller 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 clause 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 behaviour defined by MGCI is an integral part of the Recommendation, 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 (clause 7). 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 Recommendation.
- 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 analogue 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), "replicate" (replcate), "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", "conference", or "replicate".

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" or "replicate" mode are discarded.
- Audio signals received in data packets through connections in "receive only", "conference", or "send/receive" mode are mixed together and then 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 Recommendation.

- Audio signals sent to and from the endpoint are mixed and transmitted over all the connections whose mode is "replicate". This SHOULD include audio signals generated by signals.
- 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 analogue access lines, the signal will be an audio signal, and the signal MUST NOT be passed on to a telephone connected to the analogue 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.

The "replicate" mode MUST at a minimum support replicating the stream from the endpoint and one other connection regardless of the encoding method used for that other connection. The "replicate" connection is however only REQUIRED to support a resulting media stream in G.711 encoding[5]. Support of the "conference" mode is optional. Please refer to Appendix IV for illustrations of mode interactions.

6.3.1 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 accordingly[6].

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

EndpointId is the identifier for the endpoint(s) in the gateway where NotificationRequest executes. The EndpointId MUST follow the rules for endpoint names specified in 6.1.1. 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.

---

[5] The "replicate" connection can, e.g. be used to support "busy line verification" with minimal resource impact on the embedded client.

[6] The new instruction would be a ModifyConnection 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. The following are examples of signals[7]:

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

Signals are divided into different types depending upon their behaviour:

- `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 Annex 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 stopped[8]. If the signal is not stopped, the signal will time out, stop and generate an "operation complete" event, about which the Call Agent may or may not have requested to be notified. If the Call Agent has asked for the "operation complete" event to be notified, the "operation complete" event sent to the Call Agent will include the name(s) of the signal(s) that timed out[9]. Signal(s) generated on a connection will include the name of that connection. Time-out signals have a default time-out value defined for them, which may be altered by the provisioning process. Also, the time-out period may be provided as a parameter to the signal. A value of zero indicates that the time-out period is infinite. A TO signal that fails after being started, but before having generated on "operation complete" event will generate an "operation failure" event, which will include the name(s) of the signal(s), that timed out[9].
- `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 mode of the connection. For example, if a call-waiting tone is applied to an endpoint involved in an active call, only the party using the endpoint in question will hear the call-waiting tone. However, individual signals may define a different behaviour.

---

7 Please refer to 0 for a complete list of signals.

8 The "Keep signal(s) active" action may override this behaviour.

9 If parameters were passed to the signal, the parameters will not be reported.

When a signal is applied to a connection that has received a RemoteConnectionDescriptor (see 6.3.3), the media stream generated by that signal MUST be forwarded on the connection *regardless* of the current mode of the connection. If a RemoteConnectionDescriptor has not been received, the gateway MUST return an error (error code 527 – missing RemoteConnectionDescriptor).

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

The currently defined signals can be found in Annex A.

RequestedEvents is a list of events that the gateway is requested to detect on the endpoint. Unless otherwise specified, events are detected on the endpoint, however some events can be detected on a connection. Examples of events are[10]:

- 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);
- DTMF digits (or pulse digits).

The currently defined events can be found in Annex 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.
- Embedded ModifyConnection.

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 action[11]. Thus, informally, persistent events can be viewed as always being implicitly included in the list of RequestedEvents with an action to Notify, although no glare detection, etc., will be performed[12]. Persistent events are identified as such through their definition – see Annex A.

Non-persistent events are those events that have to be explicitly included in the RequestedEvents list. The (possibly empty) list of requested events completely replaces the previous list of requested events. In addition to the persistent events, only the events specified in the requested events list will be detected by the endpoint. If a persistent event is included in the RequestedEvents list, the action

---

[10] These are merely examples from the example line package in Appendix I.

[11] Thus the RequestIdentifier will be the RequestIdentifier of the current NotificationRequest.

[12] 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.

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:

	Notify	**Accumu-late**	**Accumu-late according to digit map**	**Ignore**	**Keep Signal(s) Active**	**Embedded Notification Request**	**Embedded Modify Connection**
Notify	–	–	–	–	√	–	√
Accumulate	–	–	–	–	√	√	√
Accumulate according to digit map	–	–	–	–	√	–	√
Ignore	–	–	–	–	√	–	√
Keep Signal(s) active	√	√	√	√	–	√	√
Embedded Notification Request	–	√	–	–	√	–	√
Embedded Modify Connection	√	√	√	√	√	√	–

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 6.1.5. 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 6.4.3.1 for further details.

The signals being applied by the SignalRequests are 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 mandated a "ringing" signal and the event request asked to look for an "off-hook" event, the ringing should, by default, stop as soon as the gateway detected an off-hook event. If "off-hook" was defined as a persistent event and the event request did not ask to look for an "off-hook" event, the ringing would stop anyway since off-hook would then be 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 behaviour would be to stop all active time-out signals when the first digit[13] 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 6.4.3.1).

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.

If connection modes are desired to be changed when a looked-for event occurs, the "Embedded ModifyConnection" action can be used. The embedded ModifyConnection may include a list of connection mode changes each consisting of the mode change and the affected connection-id. The wildcard "$" can be used to denote "the current connection", however this notation MUST NOT be used outside a connection handling command – the wildcard refers to the connection in question for the connection handling command.

The embedded ModifyConnection action allows the Call Agent to instruct the endpoint to change the connection mode of one or more connections immediately following the detection of the associated event. Each of connection mode changes work similarly to a corresponding ModifyConnection command[14]. When a list of connection mode changes is supplied, the connection mode changes MUST be applied one at a time in left-to-right order. When all the connection mode changes have finished, an "operation complete" event parameterized with the name of the completed action will be generated (see Annex A for details). Should any of the connection mode changes fail, an "operation failure" event parameterized with the name of the failed action and connection mode change will be generated (see Annex A for details) – the rest of the connection mode changes MUST NOT be

---

[13] Digit as defined in digit maps, i.e. including asterisk, timer, etc.

[14] Thus, if, e.g. D-QoS is used on the connection, the default D-QoS action will still be taken when the embedded ModifyConnection action is carried out.

attempted, and the previous successful connection mode changes in the list MUST NOT be changed either.

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.

Clause 6.4.3.1 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 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 Annex 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 exceptions to this rule, for example, fax and modem tones, which can be detected but can not be signalled. 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 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, e.g. ringback, should stop.

QuarantineHandling is an optional parameter that specifies handling options for the quarantine buffer (see 6.4.3.1). 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 6.4.3.1.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 6.5) optionally followed by commentary.

6.3.2 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 6.1.1. 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. Events that were detected on a connection will include the name of that connection. 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 6.5) optionally followed by commentary.

6.3.3 CreateConnection

This command is used to create a connection.

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

This function is used when setting up a connection between two endpoints. A connection is defined by its attributes and the endpoints it associates. The input parameters in CreateConnection provide the data necessary to build one of the two endpoints "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 send and receive characteristics of the media connection. The basic fields contained in LocalConnectionOptions are:

- **Encoding Method**: A list of literal names for the compression algorithm (encoding/decoding method) used to send and receive media on the connection MUST be specified with at least one value. The entries in the list are ordered by preference. The endpoint MUST choose exactly one of the codecs, and the codec SHOULD be chosen according to the preference indicated. If the endpoint receives any media on the connection encoded with a different encoding method, it MAY discard it. The endpoint MUST additionally indicate which of the remaining compression algorithms it is willing to support as alternatives – see 7.4.1 for details. A list of permissible encoding methods is specified in a separate IPCablecom 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 is specified in a separate IPCablecom document.
- **Echo Cancellation**: Whether echo cancellation should be used on the line side or not[15]. 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 $A0_H$ applies corresponding to an IP precedence bits setting of five.
- **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 following LocalConnectionOptions fields are used to support Dynamic Quality of Service (D-QoS) (please refer to Annex B for further details):

- **D-QoS GateID**: The GateID for the gate that has been set up at the edge router. The Gate-ID is a 32-bit identifier encoded as a string of up to 8 hex characters. This parameter is optional in general, but mandatory when D-QoS resource reservation and/or committal is to be performed. The presence of this parameter implies that D-QoS is to be performed for this command, where as absence implies that D-QoS is not to be performed.
- **D-QoS Resource Reservation**: Allows explicit control over whether D-QoS resource reservation and/or committal should be performed in the send and/or receive direction or not. The parameter is optional and can have one or more of the following values:

---

[15] Echo cancellation on the packet side is not supported.

Reserve values:

- "SendReserve" Resources are reserved in the send direction only.
- "ReceiveReserve" Resources are reserved in the receive direction only.
- "SendReceiveReserve" Resources are reserved in the send and receive directions.

Commit values:

- "SendCommit" Resources are committed in the send direction only.
- "ReceiveCommit" Resources are committed in the receive direction only.
- "SendReceiveCommit" Resources are committed in the send and receive directions.

The parameter is optional, and multiple values are separated by commas. When D-QoS is to be performed, and the parameter is either omitted or no value is present, resource reservation MUST be performed for both the send and receive directions. The resources reserved are determined by the coding parameters applied to the connection, i.e. encoding method, packetization period, silence suppression, ciphersuite, etc. External parameters, such as the use of payload header suppression may affect the amount of resources reserved as well – please see the IPCablecom Dynamic Quality of Service Specification ITU-T J.163 for details.

Receive resources can be reserved and committed without having obtained a RemoteConnectionDescriptor, whereas send resources can be reserved, but not committed, until a RemoteConnectionDescriptor is supplied. When D-QoS reservation is to be performed, and the parameter is either omitted or no value is present, resources MUST by default be committed based on the connection mode as specified in the table below:

Connection Mode	D-QoS
"inactive"	Do not commit
"send only", "replicate"	Commit send
"receive only"	Commit receive
"send/receive", "conference", "network loopback", "network continuity test"	Commit send and receive

If a different commit operation is desired, the appropriate commit value is supplied and will be used instead. If a commit operation is to be performed, but no reservation has been made, or an existing reservation does not fully satisfy the resources to be committed[16], a reservation will be made automatically. If a reserve value is specified, but no commit value is specified, a commit operation will not be performed.

- **ResourceID**: An existing ResourceID for resources already reserved at the edge router. The use of the ResourceID allows separate reservations to reserve the same resource, however only one of the reservations can be active at a given point in time. The ResourceID is a 32-bit identifier encoded as a string of up to 8 hex characters. The parameter is optional.
- **ReserveDestination**: This optional parameter may specify an IPv4 address, optionally followed by a colon and a UDP port number, that is the destination for the resource reservation. When a UDP port number is not specified, a default value of 9 applies. The ReserveDestination is typically used when resource reservation is to be performed, and a RemoteConnectionDescriptor has not yet been provided for the connection. This enables reservations and downstream commits to be sent to the edge router when the source of a

[16] This is not possible for the CreateConnection command but is noted here for completeness. It is possible for the ModifyConnection command, however (see 6.3.4).

media stream is not yet known[17]. When a RemoteConnectionDescriptor has been provided, the parameter is ignored.

The following LocalConnectionOptions fields are used to support the IPCablecom security services:

- **Secret**: The optional secret is a seed value that MUST be used to derive end-to-end encryption keys for the RTP and RTCP security services as specified in the IPCablecom Security specification (under development). The secret SHOULD be encoded as clear-text if it only contains values in the ASCII character range 21_H to $7E_H$. Otherwise, the secret MUST be encoded using base64 encoding. If no value is supplied, or the parameter is omitted and security services are to be used, the endpoint MUST generate a secret on its own[18]. When a secret is supplied by the CA, the secret SHOULD be used.
- **RTP ciphersuite**: A list of ciphersuites for RTP security in order of preference. The entries in the list are ordered by preference where the first ciphersuite is the preferred choice. The endpoint MUST choose exactly one of the ciphersuites. The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives (see 7.4.1 for details). Each ciphersuite is represented as ASCII strings consisting of two (possibly empty) substrings separated by a slash ("/"), where the first substring identifies the authentication algorithm, and the second substring identifies the encryption algorithm. A list of permissible ciphersuites are specified in the ITU-T J.170 IPCablecom Security Specification (under development).
- **RTCP ciphersuite**: A list of ciphersuites for RTCP security in order of preference. The entries in the list are ordered by preference where the first ciphersuite is the preferred choice. The endpoint MUST choose exactly one of the ciphersuites. The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives. See 7.4.1 for details. Each ciphersuite is represented as ASCII strings consisting of two (possibly empty) substrings separated by a slash ("/"), where the first substring identifies the authentication algorithm, and the second substring identifies the encryption algorithm. A list of permissible ciphersuites are to be specified in the ITU-T J.170 IPCablecom Security Specification, which is under development.

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. Clause 7.4 details the supported use of SDP in the NCS profile. This parameter may have a null value when the information for the remote 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.

The NCS profile currently assumes that the same media parameters apply to a connection in both the send and receive directions. Part of the information in the RemoteConnectionDescriptor is therefore redundant and a potential for inconsistency with the LocalConnectionOptions exists. It is however purely the responsibility of the Call Agent to ensure that it issues coherent commands to each endpoint to ensure that consistent media parameters are specified. If inconsistency is detected by a gateway though, the LocalConnectionOptions will simply take precedence. When codecs are

---

[17] Note that this will enable certain theft-of-service scenarios. See the Dynamic Quality of Service Specification (ITU-T J.163) for details.

[18] This includes both generating a new secret and using a secret supplied in a RemoteConnectionDescriptor.

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", "replicate", "network loopback" or "network continuity test". The handling of these modes is specified in the beginning of Section 6.3. 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", "replicate", 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, e.g. addresses and RTP ports for "IN" connections as defined in SDP. It is similar to the RemoteConnectionDescriptor, except that it specifies this side of the connection. Clause 7.4 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 behaviour of the gateway is determined by the value of the mode parameter (subject to security):

- 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 (as always) 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", "replicate", 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 cancelling 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 6.5) optionally followed by commentary.

ResourceID is a D-QoS parameter that may be returned by the gateway. When a successful D-QoS resource reservation is made, the ResourceID provides a handle for the resources reserved.

6.3.4 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]
     [, ResourceID]
               ← 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 details of D-QoS operation were specified in the CreateConnection command and generally the same rules apply here, except as noted below:

- **D-QoS GateID**: A D-QoS GateID is mandatory when D-QoS operation is required, unless D-QoS operation has previously been done for the connection in question. In the latter case, the previously supplied D-QoS GateID will then be used.
- **D-QoS Resource Reservation**: Allows explicit control over whether D-QoS resource reservation and/or committal should be performed in the send and/or receive direction or not. The parameter is optional and multiple values can be specified. When the parameter is omitted and D-QoS reservation is to be performed, the default is to reserve in both the send and receive directions, unless a suitable reservation for the connection has already been made (see Annex B). In that case, a new reservation will not be made. Resources are committed the same way as for CreateConnection, except when changing to "inactive" mode. In that case, the committed resources MUST be lowered to zero. An existing resource reservation is still maintained though.
- **ResourceID**: The parameter is optional. When supplied, it replaces the ResourceID kept by the embedded client for the connection.
- **ReserveDestination**: The parameter is optional. When supplied, it replaces the ReserveDestination kept by the embedded client for the connection. If a RemoteConnectionDescriptor has been supplied for the connection, the parameter is ignored.

The command will only return a **LocalConnectionDescriptor** if the local connection parameters, such as, e.g. RTP ports, etc. 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 old value of that parameter will be retained if possible. If a parameter change necessitates a change in one or more *unspecified* parameters, the gateway is free to choose suitable values for the unspecified parameters that must change[19].

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 Agent[20]. 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 ITU-T J.170 IPCablecom Security Specification (under development) should be consulted for additional security requirements.

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

---

[19] This can for instance happen if a codec change is specified, and the old codec used silence suppression, but the new one does not support it. If, e.g. the packetization period furthermore was not specified, and the new codec supported the old packetization period, the value of this parameter would not change, as a change would not be necessary.

[20] For instance, if media needs to traverse a firewall.

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 6.5) optionally followed by commentary.

ResourceID is a D-QoS parameter that is returned by the gateway if it performs a resource reservation and obtains a new ResourceID from the edge router. When a successful D-QoS resource reservation is made, the ResourceID provides a handle for the resources reserved.

6.3.5 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]
                        [, 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. When one or more D-QoS reservations and/or committals have been made for the connection, the DeleteConnection command will release the resources reserved.

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, 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 6.5) optionally followed by commentary.

6.3.6 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 6.6.

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 DeleteConnection. When one or more D-QoS reservations and/or committals have been made for the connection, the embedded client will release the resources reserved.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number (see 6.5) optionally followed by commentary.

6.3.7 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 6.1.1. 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. When one or more D-QoS reservations and/or committals have been made for the connection, the embedded client will release the resources reserved.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 6.5) optionally followed by commentary.

6.3.8 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 voice 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 IPCablecom 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 Recommendation.

6.3.8.1 AuditEndPoint

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

```
{ ReturnCode
          [, EndPointIdList]
          [, NumEndPoints] } |
{ ReturnCode
          [, RequestedEvents]
          [, DigitMap]
          [, SignalRequests]
          [, RequestIdentifier]
          [, NotifiedEntity]
          [, ConnectionIdentifiers]
          [, DetectEvents]
          [, ObservedEvents]
          [, EventStates]
          [, Capabilities] }
     ← AuditEndPoint(EndpointId
                                [, RequestedInfo] |
                           [, SpecificEndPointID]
                                    [, MaxEndPointIDs] })
```

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 – RequestedInfo MUST NOT be included in this case. **MaxEndPointIDs** is a numerical value that indicates the maximum number of EndpointIds to return. If additional endpoints exist, the **NumEndPoints** return parameter MUST be present and indicate the total number of endpoints that match the EndpointID specified. In order to retrieve the next block of EndpointIDs, the **SpecificEndPointID** is set to the value of the last endpoint returned in the previous EndPointIDList, and the command is issued.

When the wildcard convention is not used, the (possibly empty) **RequestedInfo** describes the information that is requested for the EndpointId specified – the SpecificEndpointID and MaxEndpointID parameters MUST NOT be used then. The following endpoint-specific information can then be audited with this command:

RequestedEvents, DigitMap, SignalRequests, RequestIdentifier, NotifiedEntity, ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, VersionSupported, 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 signals[21]. Time-Out signals that have timed-out, and currently playing Brief signals are not included. Parameterized signals are reported with the parameters they were applied with.
- **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.
- **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 in the example line package 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.
- **VersionSupported** – A list of protocol versions supported by the endpoint.
- **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. If an endpoint is queried about a capability it does not understand, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response:
- **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.

---

[21] Currently, there should be no pending brief signals.

- **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.
- **Dynamic Quality of Service** – Whether Dynamic Quality of Service is supported or not.
- **Security** – Whether IPCablecom Security services are supported or not. If supported, the following parameters may be present as well.
- **RTP Ciphersuites** – A list of authentication and encryption algorithms supported for RTP.
- **RTCP Ciphersuites** – A list of authentication and encryption algorithms supported for RTCP.

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 6.5) 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.

6.3.8.2 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.

- **RemoteConnectionDescriptor** – The RemoteConnectionDescriptor that was supplied to the gateway for the connection.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 6.5) optionally followed by commentary.

If no information was requested, and the EndpointId refers to a valid endpoint, the gateway simply checks that the connection specified exists and, if so, returns a positive response (return code 200 – transaction executed).

6.3.9 Restart in Progress

The RestartInProgress command is used by the gateway to signal that an endpoint, or a group of endpoints, is taken out of service or is being placed back in service.

```
ReturnCode
[, NotifiedEntity]
[, VersionSupported]
              ← RestartInProgress(EndpointId
                                  , RestartMethod
                                  [, RestartDelay])
```

The **EndpointId** identifies the endpoints that are taken in or out of service. The "all of" wildcard convention can be used to apply the command to a group of endpoints, for example, all endpoints that are attached to a specified interface, or even all endpoints that are attached to a given gateway. The "any of" wildcard convention MUST NOT be used.

The RestartMethod parameter specifies the type of restart:

- A "graceful" restart method indicates that the specified endpoint(s) will be taken out of service after the specified "restart delay". The established connections are not yet affected, but the Call Agent should refrain from establishing new connections, and should try to gracefully tear down any existing connections.
- A "forced" restart method indicates that the specified endpoints are taken out of service abruptly. The established connections, if any, are lost.
- A "restart" method indicates that service will be restored on the endpoints after the specified "restart delay". There are no connections that are currently established on the endpoints.
- A "disconnected" method indicates that the endpoint has become disconnected and is now trying to establish connectivity. The "restart delay" specifies the number of seconds the endpoint has been disconnected. Established connections are not affected.

The optional "restart delay" parameter is expressed as a number of seconds. If the number is absent, the delay value should be considered null. In the case of the "graceful" method, a null delay indicates that the Call Agent should simply wait for the natural termination of the existing connections, without establishing new connections. The restart delay is always considered null in the case of the "forced" 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. To mitigate the effects of a client IP address change, the Call Agent MAY wish to resolve the embedded client's domain name by querying the DNS regardless of the TTL of a current resource record for the restarted embedded client.

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 shut down, 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 6.4.3.5 – 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 6.4.3.6. 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 wildcarding 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 6.5) 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.

Finally, a **VersionSupported** parameter with a list of supported versions may be returned if the response indicated version incompatibility (error code 528).

6.4 States, failover and race conditions

In order to implement proper call signalling, 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.

6.4.1 Recaps and highlights

As mentioned in 6.1.4, 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 clause 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 wildcarded 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 explicitly[22], 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.

[22] This could for instance happen by specifying an empty NotifiedEntity parameter.

- 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 piggybacked 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.

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

6.4.2 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 7.5), 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 Tt_{hist} that specifies the number of seconds that responses to old transactions must be kept for. The default value for Tt_{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:

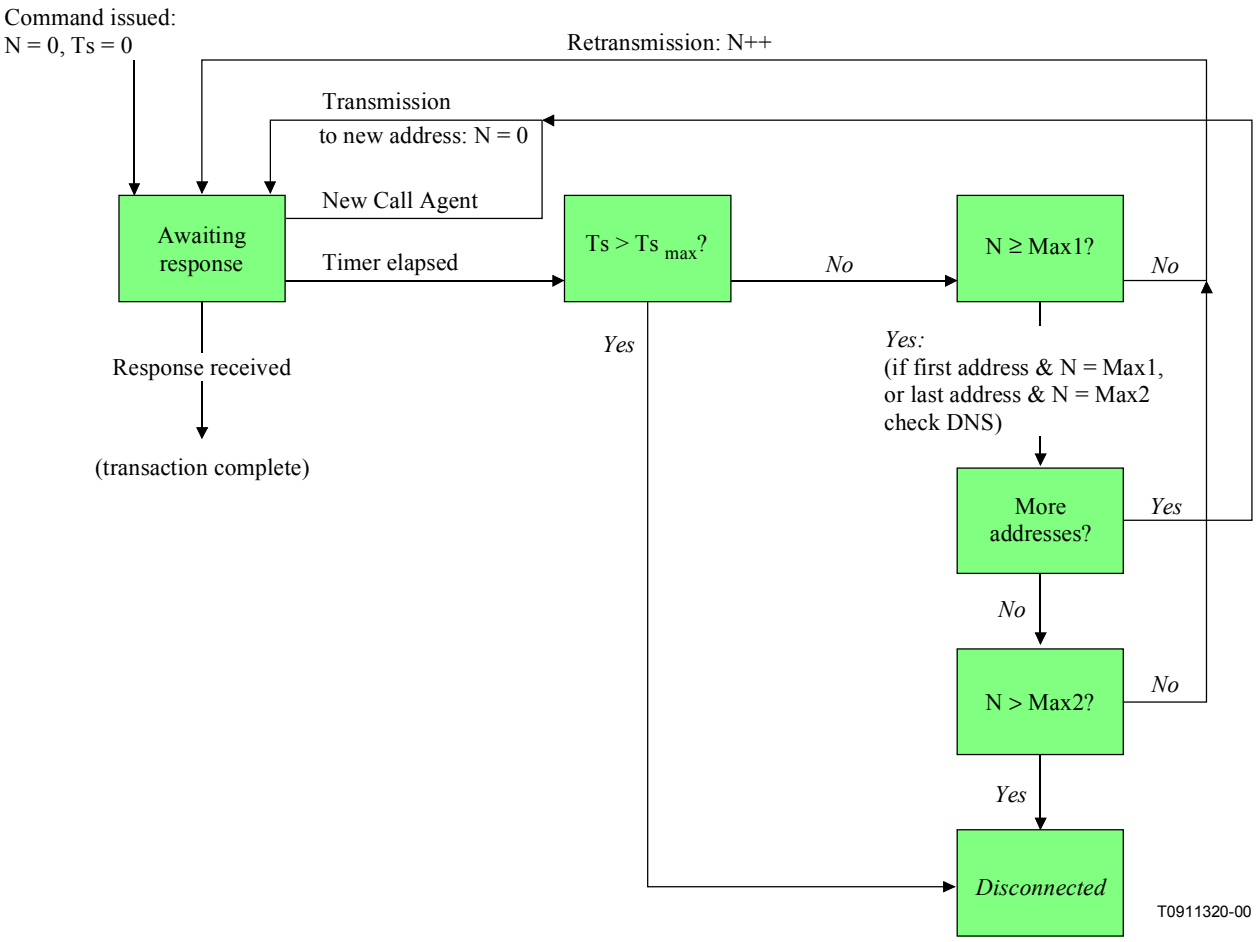
- transmission errors when, e.g. a packet is lost due to noise on a line or congestion in a queue;
- component failure when, e.g. an interface for a call agent becomes unavailable;
- call agent failure when, e.g. all interfaces for a call agent becomes unavailable;
- failover, when a new call agent is "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 1000 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 for the endpoint(s) redirected to that new Call Agent. Responses to new or old commands are still transmitted to the source address of the command.
- Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than Ts_{max}. If more than Ts_{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 6.4.3.6.



In order to adapt to network load automatically, MGCP specifies exponentially increasing timers (see 7.5.2). 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 Ts_{max}, Tt_{hist}, and the maximum transit time, Tp_{max}. Specifically, the following relation MUST be satisfied to prevent retransmitted commands from being executed more than once:

$$Tt_{hist} \geq Ts_{max} + Tp_{max}$$

The default value for Ts_{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.

6.4.3 Race conditions

In this clause 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 behaviour 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 re-established, many gateways may decide to send RestartInProgress commands simultaneously, which could lead to very unstable operation if not carefully controlled.

6.4.3.1 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 only consists of persistent events for the endpoint, and the digit map is 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 received[23]. The Notify command may be retransmitted in the "notification state", as specified in 6.4.2.

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".

---

[23] It should be noted that the Notify action cannot be combined with an Embedded NotificationRequest.

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 "piggybacking" 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;

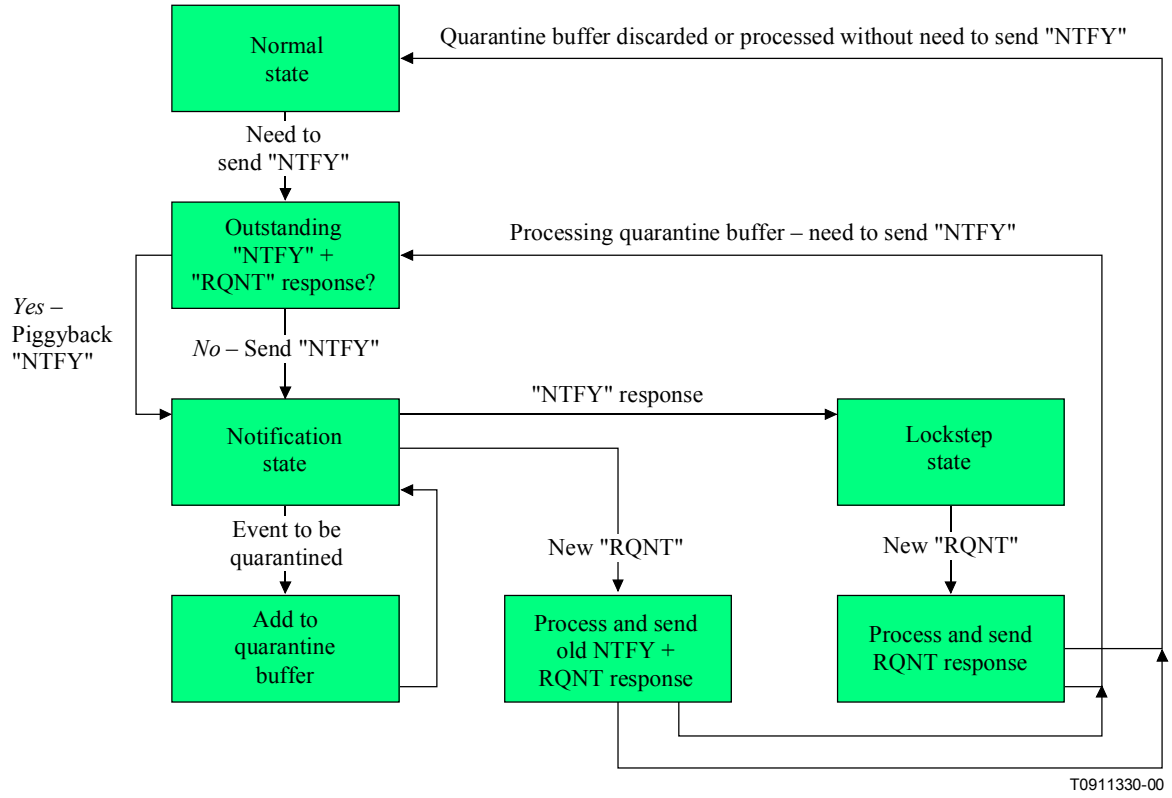
3) a copy of the outstanding Notify command is kept until a response is received. If a time-out occurs, the Notify will be repeated, in a packet that will also carry a repetition of the response to the NotificationRequest.

   - 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.
   - If the gateway has to transmit a new Notify before a response to the previous Notify is received, it constructs a packet that piggybacks 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 behaviour 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 piggybacking mechanism[24].

6.4.3.2 Explicit detection

A key element of the state of several endpoints is the position of the hook. Race conditions and state mismatch may 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 voice-based capabilities).

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).

Additionally, individual signal definitions can specify that a signal will only operate under certain conditions, e.g. ringing may only be possible if the phone is already off-hook. If such prerequisites exist for a given signal, the gateway MUST return the error specified in the signal definition if the prerequisite is not met.

It should be noted, that the condition check is performed at the time the notification request is received, where as the actual event that caused the current condition may have either been reported, or ignored earlier, or it may currently be quarantined.

---

[24] Vendors that choose not to follow this Recommendation should examine Call Agent failure scenarios carefully.

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.

6.4.3.3 Transactional semantics

As the potential transaction completion times increases, e.g. due to external resource reservations, a careful definition of the transactional semantics becomes increasingly important. In particular the issue of race conditions, specifically as it relates to hook-state must be defined carefully.

An important point to consider is, that the hook-state may in fact change between the time a transaction is initiated and the time it completes. More generally, we may say that the successful completion of a transaction depends on one or more pre-conditions where one or more of the pre-conditions may change dynamically during the execution of the transaction.

The simplest semantics for this is simply to require that all pre-conditions MUST be met from the time the transaction is initiated until the transaction completes. Thus, if any of the preconditions change during the execution of the transaction, the transaction MUST fail. Furthermore, as soon as the transaction is initiated, all new events are quarantined. When the outcome of the transaction is known, all quarantined events are then processed.

As an example, consider a transaction that includes a request for the "off-hook" event. When the transaction is initiated, the phone is "on-hook" and this pre-condition is therefore met. If the hook-state changes to "off-hook" before the transaction completes, the pre-condition is no longer met, and the transaction therefore immediately fails. The "off-hook" event will now be stored in the "quarantine" buffer which then gets processed.

6.4.3.4 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 wildcarded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The Call Agent should individually delete all connections whose completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wildcarding should not be sent until a response to the wildcarded 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 are not subject to any sequencing.

8) RestartInProgress must always be the first command sent by an endpoint as defined by the restart procedure (see 6.4.3.5). Any other command or response must be delivered after this RestartInProgress command (piggybacking allowed).

9) When multiple messages are piggybacked in a single packet, the messages are always processed in order.

Those of the above rules that specify gateway behaviour 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.

6.4.3.5 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 behaviour 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 piggybacking 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 piggybacked 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.

Should the gateway enter the "disconnected" state while carrying out the restart procedure, the disconnected procedure specified in 6.4.3.6 MUST be carried out, except that a "restart" rather than "disconnected" message is sent during the procedure.

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 wildcarding 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.

6.4.3.6 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 6.4.2. 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 piggybacking in achieving this. The Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint.

This Recommendation purposely does not specify any additional behaviour 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 on affected endpoints.

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.

6.5 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:

- value 000 indicates a response acknowledgement[25];
- values between 100 and 199 indicate a provisional response;
- values between 200 and 299 indicate a successful completion;
- values between 400 and 499 indicate a transient error;
- values between 500 and 599 indicate a permanent error.

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

Code	Meaning
000	Response acknowledgement.
100	The transaction is currently being executed. An actual completion message will follow later.
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.

[25] Response acknowledgement is used for provisional responses (see 7.8).

Code	Meaning
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.
529	Internal hardware failure.
532	Unsupported value(s) in LocalConnectionOptions.
533	Response too big.

6.6 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).
903	QoS resource reservation was lost.

7 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 6.3.6. 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.

7.1 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.

7.2 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.

7.2.1 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.

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

An embedded client that receives a command with an experimental verb it does not support MUST return an error (error code 511 – unrecognized extension).

7.2.1.2 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 7.5). 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 Recommendation though.

Transaction identifiers for all commands sent from a given embedded client MUST be unique for the maximum lifetime of the transactions (see 7.5) 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 embedded client in turn can always detect a duplicate response acknowledgement by looking at the transaction id(s).

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.

7.2.1.3 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	Analogue 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 6.1.1.

7.2.1.4 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 Recommendation is 1.0.

The protocol version for this Recommendation MUST be encoded as:

```
MGCP 1.0 NCS 1.0
```

The "NCS 1.0" portion signals that this is the NCS 1.0 profile of MGCP 1.0.

An entity that receives a command with a protocol version it does not support, MUST respond with an error (error code 528 – Incompatible Protocol Version).

7.2.2 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
ResponseAck[26]	K	See description.
CallId	C	Hexadecimal string; length MUST NOT exceed 32 characters.
ConnectionId	I	Hexadecimal string; length MUST NOT exceed 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	Hexadecimal string; length MUST NOT exceed 32 characters.
LocalConnectionOptions	L	See description.

[26] The ResponseAck parameter was not shown in 6.3 as transaction identifiers are not visible in our example API. Implementers may choose a different approach.

Parameter name	Code	Parameter value
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.
MaxEndPointIds	ZM	Decimal string; length MUST NOT exceed 16 characters.
NumEndPoints	ZN	Decimal string; length MUST NOT exceed 16 characters.
RequestedInfo	F	See description.
QuarantineHandling	Q	See description.
DetectEvents	T	See description.
EventStates	ES	See description.
ResourceID	DQ-RI	See description.
RestartMethod	RM	See description.
RestartDelay	RD	A number of seconds encoded as a decimal number.
Capabilities	A	See description.
VersionSupported	VS	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
ResponseAck[26]	O	O	O	O	O	O	O	O
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
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	O	F	F

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
MaxEndPointIds	F	F	F	F	F	O	F	F
NumEndPoints	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
ResourceID	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
VersionSupported	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. For the connection handling commands, this applies as well when a RequestIdentifier is included.								

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 MUST 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 X*ABC*, 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.

7.2.2.1 Response acknowledgement

The response acknowledgement parameter[26] is used to support the three-way handshake described in 7.7. It contains a comma-separated list of "confirmed transaction-id ranges."

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

An example of a response acknowledgement is:

```
K: 6234-6255, 6257, 19030-19044
```

7.2.2.2 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest that triggered it. A RequestIdentifier is a hexadecimal string; length MUST NOT exceed 32 characters. The string "0" is reserved for reporting of persistent events in the case where no NotificationRequest has been received yet (see 6.3.2).

7.2.2.3 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 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".

The LocalConnectionOptions parameters used for Dynamic Quality of Service are:

- the D-QoS GateID encoded as the keyword "dq-gi" followed by a colon and a string of up to 8 hex characters corresponding to a 32-bit identifier for the GateID;
- the D-QoS Resource Reservation parameter encoded as the keyword "dq-rr" 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 possible values are:

Mode	Meaning
sendresv	Reserve in the send direction only.
recvresv	Reserve in the receive direction only.
snrcresv	Reserve in the send and receive directions.
sendcomt	Commit in the send direction only.
recvcomt	Commit in the receive direction only.
snrccomt	Commit in the send and receive directions.

- the ResourceID encoded as the keyword "dq-ri" followed by a colon and a string of up to 8 hex characters corresponding to a 32-bit identifier for the ResourceID;
- the ReserveDestination is encoded as the keyword "dq-rd" followed by a colon and an IP-address encoded similarly to an IP-address for the domain name portion of an endpoint name. The ReserveDestination may optionally be followed by a colon and up to 5 decimal characters for a UDP port number to use.

The LocalConnectionOptions parameters used for Security are encoded as follows:

- the secret is encoded as the keyword "sc-st" followed by a colon, a method, a colon, and the actual secret. The method is either the string "clear" if the secret is encoded in clear-text, or the string "base64" if the secret is encoded using base64;
- the RTP ciphersuite is encoded as the keyword "sc-rtp" followed by a colon and an RTP ciphersuite string as defined below. A list of values may be specified in which case the values will be separated by a semicolon;

- the RTCP ciphersuite is encoded as the keyword "sc-rtcp" followed by a colon and an RTCP ciphersuite string as defined below. A list of values may be specified in which case the values will be separated by a semicolon.

The RTP and RTCP ciphersuite strings follow the grammar:

```
ciphersuite =              [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]
AuthenticationAlgorithm =  1*( ALPHA / DIGIT / "-"/ "_" )
EncryptionAlgorithm =      1*( ALPHA / DIGIT | "-" / "_" )
```

where ALPHA and DIGIT are defined in RFC 2234. White spaces are not allowed within a ciphersuite. The following example illustrates the use of ciphersuite:

62/51

The actual list of IPCablecom supported ciphersuites to be provided in the ITU-T J.170 IPCablecom Security Specification.

When several parameters are present, the values are separated by a comma. It MUST be 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:20, s:on
L: p:30, a:G729A, e:on, t:A0, s:off
```

The type of service hex value "20" implies an IP precedence of 1, and a type of service hex value of "A0" implies an IP precedence of 5.

This set of attributes may be extended by extension attributes. Extension attributes are composed of an attribute name, followed by a colon, and a semicolon-separated list of attribute values. The attribute name MUST start with the two characters "x+", for a mandatory extension, or "x–", for a non-mandatory extension. If a gateway receives a mandatory extension attribute that it does not recognize, it MUST reject the command with an error (error code 525 – Unknown extension in LocalConnectionOptions).

7.2.2.4 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 can also contain a list of supported packages, and a list of supported modes.

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 indicate 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 7.2.2.7;
- the keyword "dq-gi" if Dynamic Quality of Service is supported;
- the keyword "sc-st" if IPCablecom Security is supported. In that case, the following keywords indicate the ciphersuites supported:
  - the keyword "sc-rtp" followed by a colon and a semicolon-separated list of RTP AuthenticationAlgorithms, a slash, and a semicolon-separated list of EncryptionAlgorithms supported;
  - the keyword "sc-rtcp" followed by a colon and a semicolon-separated list of RTCP AuthenticationAlgorithms, a slash, and a semicolon-separated list of EncryptionAlgorithms supported.

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

Examples of capabilities are:

```
A:   a:PCMU;G729A, p:10-100, e:on, s:off, v:L;S,
     m:sendonly;recvonly;sendrecv;inactive
A:   a:G729A; p:30-90, e:on, s:on, v:L;S,
     m:sendonly;recvonly;sendrecv;inactive;confrnce,
     dq-gi, sc-st, sc-rtp: 00/51;03
```

Note that the codecs and security algorithms are merely examples – separate IPCablecom specifications detail the actual codecs and algorithms supported, as well as the encoding used.

7.2.2.5 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

Extension connection parameter names are composed of the string "X–" followed by a two-letter extension parameter name. Call Agents that receive unrecognized extensions MUST silently ignore these extensions.

An example of a connection parameter encoding is:

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

7.2.2.6 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 6.6.

7.2.2.7 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: replcate	The gateway should only send packets according to replicate mode
M: netwloop	The gateway should place the endpoint in Network Loopback mode
M: netwtest	The gateway should place the endpoint in Network Continuity Test mode

7.2.2.8 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. The event name can optionally be followed by an "at" sign (@) and the identifier of a connection on which the event should be observed. 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.

The following are examples of event names:

X/hu	On-hook transition, in the example line package
X/0	Digit 0 in the example line package
hf	Flash-hook, assuming that the example line package is the default package for the endpoint
X/rt@0A3F58	Ringback on connection "0A3F58"

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

X/[0-9]	Digits 0 to 9 in the example line package
X/X	Digits 0 to 9 in the example line package
[0-9*#A-D]	All digits and letters in the example line package (default for endpoint)
X/all	All events in the example line package

Finally, the star sign can be used to denote "all connections", and the dollar sign can be used to denote the "current" connection. The following are examples of such notations:

X/rt@*	Ringback on all connections for the endpoint
X/rt@$	Ringback on the current connection

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

7.2.2.9 RequestedEvents

The RequestedEvents parameter provides the list of events that have been requested. The currently defined event codes are described in Annex 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 6.3.1 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
Embedded ModifyConnection	C

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, "oc" and "oc(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 (using the example line package):

```
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( <SignalRequests>) )
```

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

```
R: hd(A, E(S(dl), R( B/oc(N), [0-9#T](D) ), D((1xxxxxxxxxx|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 time-out has occurred. It should be noted, that since on-hook is a persistent event, it will still be detected and notified although it has not been specified here.

The embedded ModifyConnection action follows the format:

```
C(M(<ConnectionMode1>( <ConnectionID1> )) , … ,
                                    M(<ConnectionModen>(ConnectionIDn )))
```

The following example illustrates the use of Embedded ModifyConnection with the example line package:

```
R: hf(A, C(M(inactive(X43DC)), M(sendrecv($)))), B/oc(N), B/of(N)
```

On hook-flash, change the connection mode of connection "X43DC" to "inactive", and then change the connection mode of the "current connection" to "send receive". Notify events on "operation complete" and "operation failure".

7.2.2.10 SignalRequests

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

Some signals can be qualified by signal parameters. When a signal is qualified by multiple signal parameters, the signal parameters are separated by commas. Each signal parameter MUST follow the format specified below (white spaces allowed):

```
signal-parameter        = signal-parameter-value | signal-parameter-name
                        "="signal-parameter-value | signal-parameter-name
                        "(" signal-parameter-list ")"
signal-parameter-list   = signal-parameter-value 0*( "," signal-parameter-
                        value )
```

where signal-parameter-value may be either a string or a quoted string, i.e. a string surrounded by two double quotes. Two consecutive double-quotes in a quoted string will escape a double-quote within that quoted string. For example, `"ab""c"` will produce the string `ab"c`.

Each signal has one of the following signal-types associated with it (see 6.3.1):

- 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 from the example line package on:

vmwi(+), vmwi

Time-out signals can be parameterized with the signal parameter "TO" and a time-out value that overrides the default time-out value. If a time-out signal is not parameterized with a time-out value, the default time-out value will be used. Both of the following will apply the ringing signal from the example line package for 6 seconds:

rg(to=6000)

rg(to(6000))

Individual signals may define additional signal parameters.

The signal parameters will be enclosed within parenthesis as shown above.

When several signals are requested, their codes are simply separated by a comma.

7.2.2.11 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. When an event is detected on a connection, the observed event will identify the connection the event was detected on using the "@<connection>" syntax. Examples of observed events using the example line package are:

```
O: hu

O: 8,2,9,5,5,5,5,T
O: hf,hf,hu
```

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 6.4.3.1 for details.

7.2.2.12 RequestedInfo

The RequestedInfo parameter contains a comma-separated list of parameter codes, as defined in the "Parameter lines" clause – clause 6.3.8 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
```

7.2.2.13 QuarantineHandling

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

```
Q: process
```

7.2.2.14 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.

7.2.2.15 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.

7.2.2.16 ResourceID

The ResourceID parameter is a return parameter used for Dynamic Quality of Service to signal the resource ID assigned for the gate in question. The ResourceID is encoded as a string of up to 8 hex characters, such as for example:

```
DQ-RI: AB345DC
```

7.2.2.17 RestartMethod

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

```
RM: restart
```

7.2.2.18 VersionSupported

The VersionSupported parameter is encoded as a comma-separated list of versions supported such as, for example:

```
VS: MGCP 1.0, MGCP 1.0 NCS 1.0
```

7.3 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
ResponseAck[26]	O*	O*	O*	O*	O*	O*	O*	O*
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
MaxEndPointIds	F	F	F	F	F	F	F	F
NumEndPoints	F	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
ResourceID	O	O	F	F	F	F	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
VersionSupported	F	F	F	F	F	O	F	O
LocalConnection Descriptor	M	O	F	F	F	F	O	F
RemoteConnection Descriptor	F	F	F	F	F	F	O	F

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

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

7.3.1 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 7.4. It is separated from the response header by an empty line, e.g.:

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
```

```
t=0 0
m=audio 3456 RTP/AVP 96
a=rtpmap:96 G726-32/8000
```

When a provisional response has been issued previously, the final response may furthermore contain the Response Acknowledgement parameter, and when Dynamic Quality of Service is used, the final response may also contain a ResourceID, as in:

```
200 1204 OK
K:
I: FDE234C8
DQ-RI: 23DB4A43

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

The final response is acknowledged by a Response Acknowledgement:

```
000 1204
```

7.3.2 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 7.4. It is separated from the response header by an empty line.

```
200 1207 OK

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

The response may also contain a ResourceID when Dynamic Quality of Service is used as in:

```
200 1207 OK
DQ-RI: 12345
```

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

```
526 1207 No bandwidth
K:
```

The final response is acknowledged by a Response Acknowledgement:

```
000 1207 OK
```

7.3.3 DeleteConnection

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in 7.2.2.5.

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

7.3.4 NotificationRequest

A NotificationRequest response does not include any additional response parameters.

7.3.5 Notify

A Notify response does not include any additional response parameters.

7.3.6 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;G728, p:10-100, e:on, s:off, t:1, v:X;B,
     m:sendonly;recvonly;sendrecv;inactive
A: a:G729A; p:30-90, e:on, s:on, t:1, v:X;B,
     m:sendonly;recvonly;sendrecv;inactive;confrnce
```

7.3.7 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;G728
M: sendrecv
P: PS=622, OS=31172, PR=390, OR=22561, PL=5, JI=29, LA=50

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

If both a local and a remote connection descriptor are provided, the local connection descriptor will be the first of the two. If a connection descriptor is requested, but it does not exist for the connection audited, that connection descriptor will appear with the SDP protocol version field only.

7.3.8 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
```

7.4 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.

7.4.1 SDP audio service use

In a voice-only gateway, we only have to describe sessions that use exactly one media, audio. The parameters of SDP that are relevant for the voice based application are specified below. Embedded clients MUST support session descriptions that conform to these rules and in the following order:

1) The SDP profile presented below.

2) RFC 2327 (SDP: Session Description Protocol).

The SDP profile provided describes the use of the session description protocol in NCS. The general description and explanation of the individual parameters can be found in RFC 2327; however, below we detail what values NCS endpoints need to provide for these fields (send) and what NCS endpoints should do with values supplied or not supplied for these fields (receive).

Any parameter not specified below SHOULD NOT be provided by any NCS endpoint, and if such a parameter is received, it SHOULD be ignored.

7.4.1.1 Protocol version (v=)

```
v= <version>
v= 0
```

Send: MUST be provided in accordance with RFC 2327 (i.e. v=0)

Receive: MUST be provided in accordance with RFC 2327.

7.4.1.2 Origin (o=)

The origin field consists (o=) of 6 sub-fields in RFC 2327:

```
o= <username> <session-ID> <version> <network-type> <address-type> <address>
o= - 2987933615 2987933615  IN   IP4     A3C47F2146789F0
```

Username:

Send: Hyphen MUST be used as username when privacy is requested. Hyphen SHOULD be used otherwise[27].

Receive: This field SHOULD be ignored.

---

[27] Since NCS endpoints do not know when privacy is requested, they SHOULD always use a hyphen.

Session-ID:

Send: MUST be in accordance with RFC 2327 for interoperability with non-IPCablecom clients.

Receive: This field SHOULD be ignored.

Version:

Send: In accordance with RFC 2327.

Receive: This field SHOULD be ignored.

Network Type:

Send: Type "IN" MUST be used.

Receive: This field SHOULD be ignored.

Address Type:

Send: Type "IP4" MUST be used

Receive: This field SHOULD be ignored.

Address:

Send: MUST be in accordance with RFC 2327 for interoperability with non-IPCablecom clients.

Receive: This field MUST be ignored.

7.4.1.3 Session name (s=)

```
s= <session-name>
s= -
```

Send: Hyphen MUST be used as Session name.

Receive: This field MUST be ignored.

7.4.1.4 Session and media information (i=)

```
i= <session-description>
```

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

7.4.1.5 URI (u=)

```
u= <URI>
```

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

7.4.1.6 E-mail address and phone number (e=, p=)

```
e= <e-mail-address>
p= <phone-number>
```

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

7.4.1.7 Connection data (c=)

The connection data consists of 3 sub-fields:

```
c= <network-type> <address-type> <connection-address>
c= IN          IP4 10.10.111.11
```

Network Type:

Send: Type "IN" MUST be used.

Receive: Type "IN" MUST be present.

Address Type:

Send: Type "IP4" MUST be used

Receive: Type "IP4" MUST be present.

Connection Address:

Send: This field MUST be filled with a unicast IP address at which the application will receive the media stream. Thus a TTL value MUST NOT be present and a "number of addresses" value MUST NOT be present. The field MUST NOT be filled with a fully-qualified domain name instead of an IP address. A non-zero address specifies both the **send and receive address for the media stream(s) it covers**.

Receive: A unicast IP address or a fully qualified domain name MUST be present. A non-zero address specifies both the send and receive address for the media stream(s) it covers.

7.4.1.8 Bandwidth (b=)

```
b= <modifier> : <bandwidth-value>
b= AS : 64
```

Send: Bandwidth information is optional in SDP but it SHOULD always be included[28]. When an rtpmap or a non well-known codec[29] is used, the bandwidth information MUST be used.

Receive: Bandwidth information SHOULD be included. If a bandwidth modifier is not included, the receiver MUST assume reasonable default bandwidth values for well-known codecs.

Modifier:

Send: Type "AS" MUST be used.

Receive: Type "AS" MUST be present.

Bandwidth Value:

Send: The field MUST be filled with the Maximum Bandwidth requirement of the Media stream in kilobits per second.

Receive: The maximum bandwidth requirement of the media stream in kilobits per second MUST be present.

---

[28] If this field is not used, the Gate Controller might not authorize the appropriate bandwidth.

[29] A non well-known codec is a codec not defined in the codec specification J.161.

7.4.1.9 Time, repeat times and time zones (t=, r=, z=)

```
t= <start-time> <stop-time>
t= 36124033 0
r= <repeat-interval> <active-duration> <list-of-offsets-from-start-time>
z= <adjustment-time> <offset>
```

Send: Time MUST be present; start time MAY be zero, but SHOULD be the current time, and stop time SHOULD be zero. Repeat Times, and Time Zones SHOULD NOT be used, if they are used it should be in accordance with RFC 2327.

Receive: If any of these fields are present, they SHOULD be ignored.

7.4.1.10 Encryption keys

```
k= <method>
k= <method> : <encryption-keys>
```

Security services for IPCablecom are to be defined by the ITU-T J.170 IPCablecom Security specification. The security services specified for RTP and RTCP do not comply with those of RFC 1889, RFC 1890, and RFC 2327. In the interest of interoperability with non-IPCablecom devices, the "k" parameter will therefore not be used to convey security parameters.

Send: MUST NOT be used.

Receive: This field SHOULD be ignored.

7.4.1.11 Attributes (a=)

```
a= <attribute> : <value>
a= rtpmap : <payload type> <encoding name>/<clock rate> [/<encoding parameters>]
a= rtpmap : 0 PCMU / 8000
a= X-pc-codecs: <alternative 1> <alternative 2> …
a= X-pc-secret: <method>:<encryption key>
a= X-pc-csuites-rtp: <alternative 1> <alternative 2> …
a= X-pc-csuites-rtcp: <alternative 1> <alternative 2> …
a= X-pc-spi-rtcp: <value>
a= X-pc-bridge: <number-ports>
a= <attribute>
a= recvonly
a= sendrecv
a= sendonly
a= ptime
```

Send: One or more of the "a" attribute lines specified below MAY be included. An attribute line not specified below SHOULD NOT be used.

Receive: One or more of the "a" attribute lines specified below MAY be included and MUST be acted upon accordingly. "a" attribute lines not specified below may be present but MUST be ignored.

rtpmap:

Send: When used, the field MUST be used in accordance with RFC 2327. It MAY be used for well-known as well as well as non well-known codecs. The encoding names used are provided in a separate IPCablecom specification.

Receive: The field MUST be used in accordance with RFC 2327.

X-pc-codecs:

Send: The field contains a list of alternative codecs that the endpoint is capable of using for this connection. The list is ordered by decreasing degree of preference, i.e. the most preferred alternative codec is the first one in the list. A codec is encoded similarly to "encoding name" in rtpmap.

Receive: Conveys a list of codecs that the remote endpoint is capable of using for this connection. The codecs MUST NOT be used until signalled through a media (m=) line.

X-pc-secret:

Send: The field contains an end-to-end secret to be used for RTP and RTCP security. The secret is encoded similarly to the encryption key (k=) parameter of RFC 2327 with the following constraints:

- The encryption key MUST NOT contain a ciphersuite, only a passphrase.
- The <method> specifying the encoding of the pass-phrase MUST be either "clear" or "base64" as defined in RFC 2045, except for the maximum line length which is not specified here. The method "clear" MUST NOT be used if the secret contains any characters that are prohibited in SDP.

Receive: Conveys the end-to-end secret to be used for RTP and RTCP security.

X-pc-csuites-rtp:
X-pc-csuites-rtcp:

Send: The field contains a list of ciphersuites that the endpoint is capable of using for this connection (respectively RTP and RTCP). The first ciphersuite listed is what the endpoint is currently expecting to use. Any remaining ciphersuites in the list represent alternatives ordered by decreasing degree of preference, i.e. the most preferred alternative ciphersuite is the second one in the list. A ciphersuite is encoded as specified below:

ciphersuite = [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]

AuthenticationAlgorithm = 1*(ALPHA / DIGIT / "-" / "_")

EncryptionAlgorithm = 1*(ALPHA / DIGIT / "-" / "_")

where ALPHA, and DIGIT are defined in RFC 2234. White spaces are not allowed within a ciphersuite. The following example illustrates the use of ciphersuite:

62/51

The actual list of ciphersuites to be provided in the ITU-T J.170 IPCablecom Security Specification.

Receive: Conveys a list of ciphersuites that the remote endpoint is capable of using for this connection. Any other ciphersuite than the first in the list cannot be used until signalled through a new ciphersuite line with the desired ciphersuite listed first.

X-pc-spi-rtcp:

Send: The field contains the IPSEC Security Parameter Index (SPI) to be used when sending RTCP packets to the endpoint for the media stream in question. The SPI is a 32-bit identifier encoded as a string of up to 8 hex characters. The field MUST be supplied when RTCP security is used.

Receive: Conveys the IPSEC SPI to be used when sending RTCP packets over IPSEC. The field MUST be present when RTCP security is used.

X-pc-bridge:

Send: NCS endpoints MUST NOT use this attribute.

Receive: NCS endpoints MUST ignore this attribute if received.

recvonly:

Send: The field MUST be used in accordance with RFC 2543.

Receive: The field MUST be used in accordance with RFC 2543.

sendrecv:

Send: The field MUST be used in accordance with RFC 2543.

Receive: The field MUST be used in accordance with RFC 2543.

sendonly:

Send: The field MUST be used in accordance with RFC 2543, except that the IP address and port number MUST NOT be zeroed.

Receive: The field MUST be used in accordance with RFC 2543.

ptime:

Send: The ptime SHOULD always be provided and when used it MUST be used in accordance with RFC 2327. When an rtpmap or non well-known codec is used, the ptime MUST be provided.

Receive: The field MUST be used in accordance with RFC 2327. When "ptime" is present, the MTA MUST use the ptime in the calculation of QoS reservations. If "ptime" is not present, the MTA MUST assume reasonable default values for well-known codecs.

7.4.1.12 Media Announcements (m=)

Media Announcements (m=) consists of 3 sub-fields:

```
M= <media> <port> <transport> <format>
M= audio 3456 RTP/AVP 0
```

Media:

Send: The "audio" media type MUST be used.

Receive: The type received MUST be "audio".

Port:

Send: MUST be filled in accordance with RFC 2327. The port specified is the receive port, regardless of whether the stream is unidirectional or bidirectional. The sending port may be different.

Receive: MUST be used in accordance with RFC 2327. The port specified is the receive port. The sending port may be different.

Transport:

Send: The transport protocol "RTP/AVP" MUST be used.

Receive: The transport protocol MUST be "RTP/AVP".

Media Formats:

Send: Appropriate media type as defined in RFC 2327 MUST be used.

Receive: In accordance with RFC 2327.

7.4.2 SDP video service use

Details on SDP use for video service are for further study.

7.5 Transmission over UDP

7.5.1 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 endpoint[30], 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 Tt_{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 time-outs for all outstanding commands and to retry commands when timeouts have been exceeded. A retransmission strategy is specified in 7.5.2.

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 6.4.

7.5.2 Retransmission strategy

This Recommendation 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 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:

- the average acknowledgement delay (AAD) estimated through an exponentially smoothed average of the observed delays;
- the average deviation (ADEV) estimated through an exponentially smoothed average of the absolute value of the difference between the observed delay and the current average.

The retransmission timer (RTO) in TCP, is set to the sum of the average delay plus N times the average deviation, where N is a constant.

---

[30] 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.

7.6 Piggybacking

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 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The piggybacked 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.

7.7 Transaction identifiers and three ways handshake

Transaction identifiers are integer numbers in the range from 1 to 999 999 999. Call Agents may decide to use a specific number space for each of the gateways that they manage, or to use the same number space for all gateways that belong to some arbitrary group. Call Agents may decide to share the load of managing a large gateway between several independent processes. These processes will share the same transaction number space. There are multiple possible implementations of this sharing, such as having a centralized allocation of transaction identifiers, or pre-allocating non-overlapping ranges of identifiers to different processes. The implementations MUST guarantee that unique transaction identifiers are allocated to all transactions that originate from any call agent sent to a particular gateway within a period of Tt_{hist} seconds. Gateways can simply detect duplicate transactions by looking at the transaction identifier only.

The Response Acknowledgement parameter can be found in any command. It carries a set of "confirmed transaction-id ranges" for final responses received – provisional responses MUST NOT be confirmed.

MGCP gateways may choose to delete the copies of the responses to transactions whose id is included in "confirmed transaction-id ranges" received in a message, however the fact that the transaction was executed MUST still be retained for Tt_{hist} seconds. Also, when a Response Acknowledgement message[31] is received, the response that is being acknowledged by it can be deleted. Gateways should silently discard further commands from that Call Agent when the transaction-id falls within these ranges, and the response was issued less than Tt_{hist} seconds ago.

Let $term_{new}$ and $term_{old}$ be the endpoint-name in respectively a new command, cmd_{new}, and some old command, cmd_{old}. The transaction-ids to be confirmed in cmd_{new} SHOULD then be determined as follows:

1) If $term_{new}$ does not contain any wildcards:
   a) Unconfirmed responses to old commands where $term_{old}$ equals $term_{new}$.
   b) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "any-of" wildcard, and the endpoint-name returned in the response was $term_{new}$.
   c) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$.
   d) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "any-of" wildcard, no endpoint-name was returned, and $term_{new}$ is covered by the wildcard in $term_{old}$.
2) If $term_{new}$ contains the "all" wildcard:
   a) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$.
3) If $term_{new}$ contains the "any of" wildcard:
   a) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$ if the "any of" wildcard in $term_{new}$ was replaced with the "all" wildcard.

A given response SHOULD NOT be confirmed in two separate messages.

The following examples illustrate the use of these rules:

- If $term_{new}$ is "aaln/1" and $term_{old}$ is "aaln/1", then the old response can be confirmed per rule 1a.
- If $term_{new}$ is "aaln/1" and $term_{old}$ is "*", then the old response can be confirmed per rule 1c.
- If $term_{new}$ is "aaln/*" and $term_{old}$ is "*", then the old response can be confirmed per rule 2a.
- If $term_{new}$ is "aaln/$" and $term_{old}$ is "aaln/*", then the old response can be confirmed per rule 3a.

The "confirmed transaction-id ranges" values SHOULD NOT be used if more than Tt_{hist} seconds have elapsed since the gateway issued its last response to that call agent, or when a gateway resumes operation. In this situation, commands should be accepted and processed, without any test on the transaction-id.

Also, a response SHOULD NOT be confirmed if the response was received more than Tt_{hist} seconds ago.

---

[31] As opposed to a command with a Response Acknowledgement parameter.

Messages that confirm responses may be transmitted and received in disorder. The gateway shall retain the union of the confirmed transaction-ids received in recent commands.

7.8 Provisional responses

In some cases, transaction completion times may be significantly longer than otherwise[32]. NCS uses UDP as the transport protocol and reliability is achieved by selective time-out-based retransmissions where the time-out is based on an estimate of the sum of the network round-trip time and transaction completion time. Significant variance in the transaction completion time is therefore problematic when rapid message loss detection without excessive overhead is desired.

In order to overcome this problem, a provisional response MUST therefore be issued if, and only if, the transaction completion time exceeds some small period of time. The provisional response acknowledges the receipt of the command although the outcome of the command may not yet be known, e.g. due to a pending resource reservation. As a guideline, a transaction that requires external communication to complete, e.g. network resource reservation, should issue a provisional response. Furthermore, if a duplicate CreateConnection or ModifyConnection command is received, and the transaction has not yet finished executing, a provisional response MUST then be sent back.

Pure transactional semantics would imply that provisional responses should not return any other information than the fact that the transaction is currently executing; however, an optimistic approach allowing some information to be returned enables a reduction in the delay that would otherwise be incurred in the system.

Provisional responses MUST only be sent in response to a CreateConnection or ModifyConnection command. In order to reduce the delay in the system, a connection identifier and session description MUST be included in the provisional response to the CreateConnection command. If a session description will be returned by the ModifyConnection command, the session description MUST be included in the provisional response here as well. If the transaction completes successfully, the information returned in the provisional response MUST be repeated in the final response. It is considered a protocol error not to repeat this information or to change any of the previously supplied information in a successful response. If the transaction fails, an error code is returned – the information returned previously is no longer valid.

A currently executing CreateConnection or ModifyConnection transaction MUST be cancelled if a DeleteConnection command for the endpoint is received. In that case, a response for the cancelled transaction SHOULD still be returned automatically, and a response for the cancelled transaction MUST be returned if a retransmission of the cancelled transaction is detected.

When a provisional response is received, the time-out period for the transaction in question MUST be set to a significantly higher value for this transaction ($Tt_{longtran}$). The purpose of this timer is primarily to detect endpoint failure. The default value of $Tt_{longtran}$ is 5 seconds; however, the provisioning process may alter this.

When the transaction finishes execution, the final response is sent and the by now obsolete provisional response is deleted. In order to ensure rapid detection of a lost final response, final responses issued after provisional responses for a transaction MUST be acknowledged. The endpoint MUST therefore include an empty "ResponseAck" parameter in those, and only those, final responses. The presence of the "ResponseAck" parameter in the final response will trigger a "Response Acknowledgement" response to be sent back to the endpoint. The "Response Acknowledgement" response will include the transaction-id of the response it acknowledges in the response header. Receipt of this "Response Acknowledgement" response is subject to the same time-out and retransmission strategies and procedures as responses to commands (see 6.4), i.e. the sender

[32] For instance when resources are reserved and committed externally as part of a transaction.

of the final response will retransmit it if the "Response Acknowledgement" is not received in time. The "Response Acknowledgment " response is never acknowledged.

8 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 to be provided in the ITU-T J.170 IPCablecom Security Specification, which should be consulted for further information.

ANNEX A

Event packages

This annex defines an initial set of event packages for the various types of endpoints currently defined by IPCablecom for embedded clients. The following packages are defined for the embedded client endpoint-types listed:

Endpoint-type	Package	Package name	Default package
Analogue Access Line	Base	B	No
Video	*For further study*	*For further study*	*For further study*
ISDN BRI	*For further study*	*For further study*	*For further study*

Each package defines a package name for the package and event codes and definitions for each of the events in the package. In the tables of events/signals for each package, there are five columns:

Code The package unique event code used for the event/signal.

Description A short description of the event/signal.

Event A check mark appears in this column if the event can be Requested by the Media Gateway Controller. Alternatively, one or more of the following symbols may appear:

"**P**" indicating that the event is persistent;

"**S**" indicating that the event is an event-state that may be audited;

"**C**" indicating that the event/signal may be detected/applied on a connection.

Signal If nothing appears in this column for an event, then the event cannot be signalled on command by the Media Gateway Controller. Otherwise, the following symbols identify the type of event:

"**OO**" On/Off signal. The signal is turned on until commanded by the Media Gateway Controller to turn it off, and vice versa.

"**TO**" Time-out signal. The signal lasts for a given duration unless it is superseded by a new signal. Default time-out values are supplied. A value of zero indicates that the time-out period is infinite. The provisioning process may alter these default values.

"**BR**" Brief signal. The event has a short, known duration.

Additional info Provides additional information about the event/signal, e.g. the default duration of TO signals.

Unless otherwise stated, all of the events/signals are detected/applied on endpoints and audio generated by them is not forwarded on any connection the endpoint may have. Audio generated by events/signals that are detected/applied on a connection will however be forwarded on the associated connection irrespective of the connection mode.

Base protocol packages

The following packages are currently defined in the base protocol. These packages apply to all endpoints:

- Base.

Base package

Package name: B

The following codes are used to identify events and signals for the "base" package for all endpoint types:

Code	Description	Event	Signal	Additional Info
oc	Operation complete	√	–	
of	Operation failure	√	–	

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 dialled 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: B/oc(mypackage/mysignal)
```

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: B/oc(mypackage/mysignal@0A3F58)
```

When the operation complete event is requested, it cannot be parameterized with any event parameters. When the package name is omitted, the default package name is assumed.

The operation complete event may additionally be generated as defined in the base protocol, e.g. when an embedded ModifyConnection command completes successfully, as in:

```
O: B/oc(B/C)
```

Operation failure (of): In general, the operation failure event may be generated when the endpoint was asked to apply one or several signals of type TO on the endpoint, and one or more of those signals failed prior to timing out. The completion report may carry as a parameter the name of the signal that failed, as in:

```
O: B/of(mypackage/mysignal)
```

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: B/of(mypackage/mysignal@0A3F58)
```

When the operation failure event is requested, event parameters can not be specified. When the package name is omitted, the default package name is assumed.

The operation failure event may additionally be generated as specified in the base protocol, e.g. when an embedded ModifyConnection command fails, as in:

```
O: B/of(B/C(M(sendrecv(AB2354))))
```

Audio

Event packages for audio is for further study.

Video

Event packages for video is for further study.

ISDN

Event packages for basic access ISDN is for further study.

ANNEX B

Dynamic Quality of Service

In this annex, we provide additional detail on the usage of Dynamic Quality of Service (D-QoS) in NCS. We describe the expected MTA behaviour in more detail and include a state machine that the MTA may implement to support the D-QoS behaviour described. The IPCablecom Dynamic Quality of Service Specification (ITU-T J.163) should be consulted for further details.

Introduction

MTA's implementing support for Dynamic Quality of Service need to store and maintain D-QoS state on a per-connection basis. Whenever D-QoS has been used for a connection, the endpoint will keep the following D-QoS information associated with the connection until it is deleted:

- **GateID** – The current GateID used for the connection.
- **ResourceID** – The current ResourceID used for the connection.
- **Last reservation** – The parameters for the most recent reservation for the connection. This includes classifiers as well as media parameters in both the send and receive direction.
- **Last commit** – The parameters for the most recent commit for the connection. This includes classifiers as well as media parameters in both the send and receive direction.
- **Reserve Destination** – An IP address and port that may be used to enable resource reservations where the remote address info is not yet known as explained below.
- **Gate Location** – The IP address and port where the D-QoS commit message should be sent to when using RSVP. The MTA learns this address through the RSVP QoS messages.

The GateID is the key to resource reservation. Once a GateID has been provided for a connection, a D-QoS state machine is created for the connection, and all of the above information will be maintained for the connection until it is either deleted, or a new GateID is provided. In the latter case, the D-QoS state machine and the above information is reset, and the old reservation is deleted[33].

Resources can be reserved and committed independently in both the send and receive direction by the MTA. The send destination IP address and port as well as the source IP address are taken from the RemoteConnectionDescriptor, when a RemoteConnectionDescriptor has been provided. In that case, the MTA MUST use the following classifiers for the resource reservation and commit:

---

[33] Note that if a ResourceID is included, and that ResourceID matches the old ResourceID, then the old reservation should not be deleted before the new one is made.

	MTA-o (J.112/RSVP)
Downstream/receive	
Source IP	IP(SDP-t)
Source Port	*
Destination IP	IP(SDP-o)
Destination Port	Port(SDP-o)
Upstream/send	
Source IP	IP(SDP-o)
Source Port	Port(o)
Destination IP	IP(SDP-t)
Destination Port	Port(SDP-t)

where:

- **IP(SDP-o)** refers to the media IP address in MTA-o's LocalConnectionDescriptor.
- **IP(SDP-t)** refers to the media IP address in MTA-o's RemoteConnectionDescriptor.
- **Port(SDP-o)** refers to the media port in MTA-o's LocalConnectionDescriptor.
- **Port(o)** refers to the source port MTA-o will be using when sending media on this connection. Note that this may or may not be the same as Port(SDP-o).

When a RemoteConnectionDescriptor has not yet been provided, the actual send destination IP address and port is unknown and the ReserveDestination address is therefore used instead. For the receive direction, the source IP address and port is wildcarded. This enables a reservation and a receive commit of the resource on the access link. The following classifiers MUST be used:

	MTA-o (J.112/RSVP)
Downstream/receive	
Source IP	*
Source Port	*
Destination IP	IP(SDP-o)
Destination Port	Port(SDP-o)
Upstream/send	
Source IP	IP(SDP-o)
Source Port	Port(o)
Destination IP	IP(RD-o)
Destination Port	Port(RD-o)

where:

- **IP(RD-o)** refers to the IP address in the ReserveDestination supplied.
- **IP(Port-o)** refers to the port number in the ReserveDestination supplied. If no port number is specified, a default value of 9 applies.
- once the actual send destination and receive source media addresses and port are known, the reservations are updated with the appropriate classifiers.
- when RSVP is used as the resource reservation protocol, the destination address used for the RSVP PATH message will be the ReserveDestination IP address supplied until a RemoteConnectionDescriptor is supplied.

NCS/D-QoS state machine

As explained above, the MTA maintains state for the Dynamic Quality of Service used on a connection. The state is derived from a state machine which is driven by the following:

- **Current state** which consists of the pair (SendQoSState, ReceiveQoSState), where each QoS state may be one of the following:
  - **N** – No resource reservation exists for the direction.
  - **R** – A resource reservation exists for the direction, but no resources are currently committed.
  - **C** – A resource reservation exists for the direction, some resources are currently committed.
  - **Connection mode** which is the NCS connection mode. The connection modes "Conference", "Network Loopback", and "Network Continuity Test" are not shown explicitly in the state machine, as they are all similar to "SendReceive". The connection mode "Replicate" is also not shown as it is similar to "SendOnly".
- **Resource Change** which is one or more of the following:
  - RemoteConnectionDescriptor IP address or port changes (classifier needs to be updated). This includes the case where it arrives for first time.
  - Codec changes.
  - Ptime changes.
  - etc.
- The **D-QoS rules** provided in 6.3.3.

As explained above, the state machine will be reinitialized when a new GateID is received. If a ResourceID is supplied as well and it is the same as the old ResourceID, the reservation(s) for the new state machine MUST be performed before the reservation(s) for the old state machine are released.

The set of possible *states* are:

- (N, N) Send resources not reserved, receive resources not reserved.
- (R, R) Send resources reserved, receive resources reserved.
- (C, R) Send resources reserved and committed, receive resources reserved.
- (R, C) Send resources reserved, receive resources reserved and committed.
- (C, C) Send resources reserved and committed, receive resources reserved and committed.
- (R, N) Send resources reserved, receive resources not reserved.
- (C, N) Send resources reserved and committed, receive resources not reserved.
- (N, R) Send resources not reserved, receive resources reserved.
- (N, C) Send resources not reserved, receive resources reserved and committed.

Once resources have been reserved and/or committed for a direction, a reservation for that direction will exist for the lifetime of the connection. The relationship between states and connection mode or D-QoS reservation parameters is shown in the table below:

	SendState	RecvState
No Reserve/Commit parameter supplied – connection mode:		
inactive	R	R
sendonly, replcate	C	R
recvonly	R	C
sendrecv, confrnce, netwloop, netwtest	C	C
Reserve/Commit parameter supplied:		
sendresv	R	N, R*
recvresv	N, R*	R
snrcresv	R	R
sendcomt	C	N, R*
recvcomt	N, R*	C
snrccomt	C	C
* If resources have been reserved or committed previously for the direction, the state will be R, otherwise the state will be N.		

The actual state transition diagram is depicted in Figure B.1:

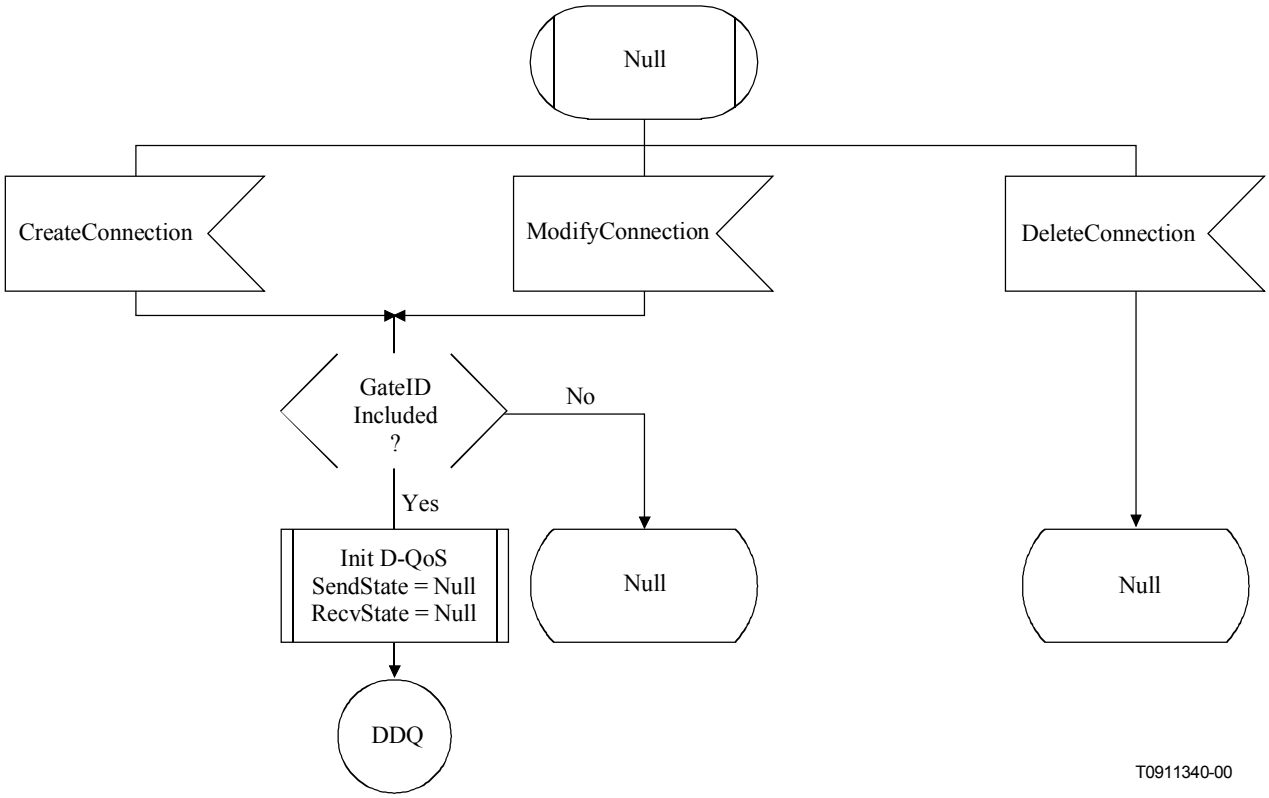


Figure B.1/J.162 – NCS/D-QoS state diagram *(sheet 1 of 2)*

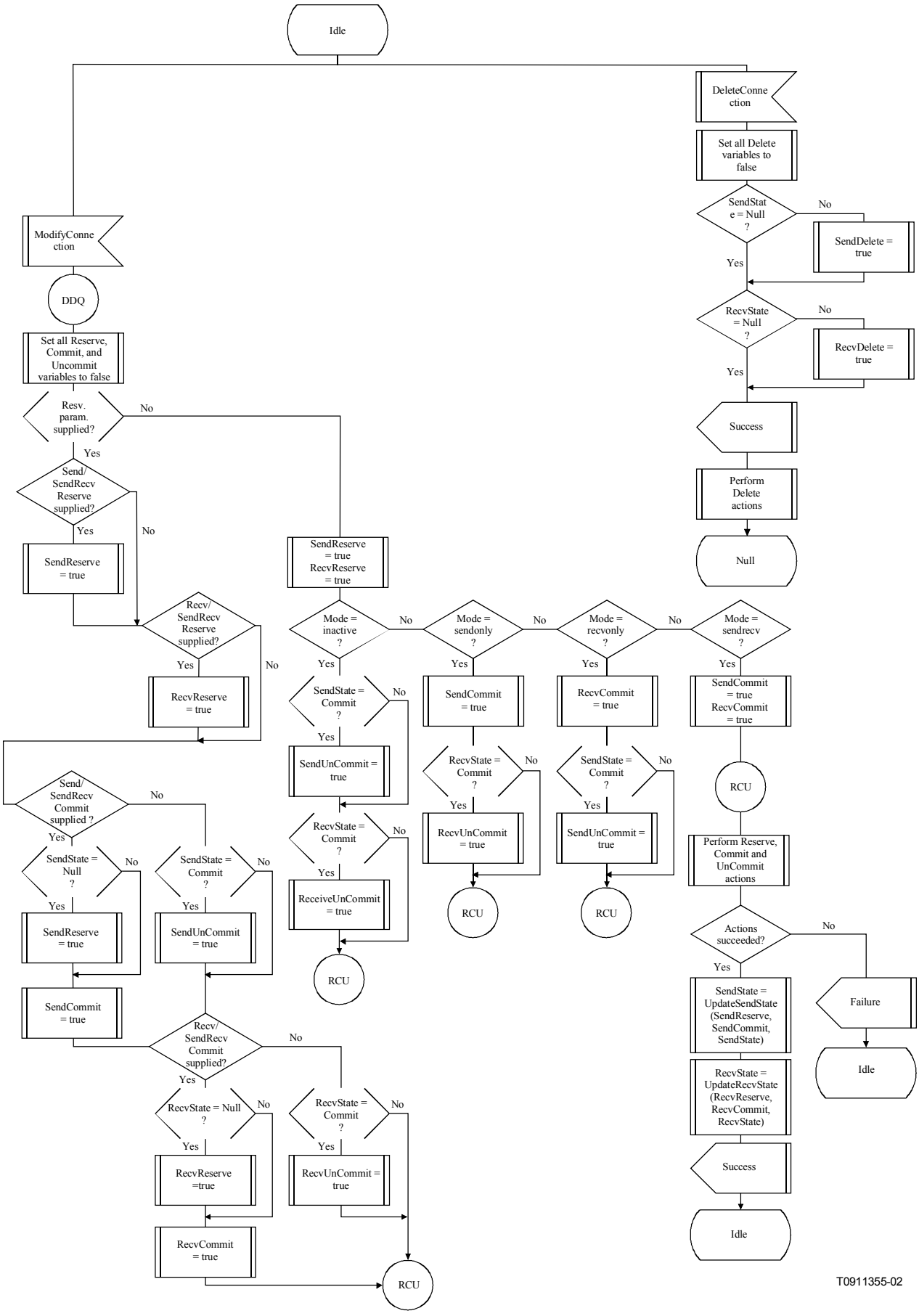


Figure B.1/J.162 – NCS/D-QoS state diagram *(sheet 2 of 2)*

When executing the state machine, boolean variables will be set to indicate whether reserve, unreserve, commit, and uncommit operations are to be performed. The pseudo-code below then provides details on individual D-QoS procedures that are to be executed as indicated by these booleans. The following *actions* specify the D-QoS actions to be taken in each of these procedures:

- **SR** means a D-QoS Send Reservation will be performed.
- **RR** means a D-QoS Receive Reservation will be performed.
- **SC** means a D-QoS Send Commit will be performed.
- **RC** means a D-QoS Receive Commit will be performed.
- **SD** means a D-QoS Send Reservation Delete will be performed.
- **RD** means a D-QoS Receive Reservation Delete will be performed.
- **SU** means a D-QoS Send Uncommit, i.e. lower committed send resources to zero, will be performed.
- **RU** means a D-QoS Receive Uncommit, i.e. lower committed send resources to zero, will be performed.

```
SendReserve()

If <current resources reserved ≠ resources to reserve> then {
                    -- skip reservation if existing reservation OK
          If <RemoteConnectionDescriptor provided> then
               SR(RemoteConnectionDescriptor)
                    -- Use RemoteConnectionDescriptor classifier
          else if <ReserveDestination provided> then
               SR(ReserveDestination)
                    -- Use ReserveDestination classifier, send to
                    -- ReserveDestination if RSVP
          else ERROR
}

ReceiveReserve()

If <current resources reserved ≠ resources to reserve> then {
                    -- skip reservation if existing reservation OK
          If <RemoteConnectionDescriptor provided> then
               RR(RemoteConnectionDescriptor)
                    -- Use RemoteConnectionDescriptor classifier
          else if <(J.112 QoS) or (RSVP and ReserveDestination provided )>
               then RR(*)
                    -- Use wildcard classifier, send to
                    -- ReserveDestination if RSVP
          else ERROR
}

SendCommit()

If <current resources committed ≠ resources to commit> then {
                    -- skip commit if existing OK
          If <RemoteConnectionDescriptor provided> then {
               If not <resources to commit ⊂ resources reserved > then {
                    -- old reservation does not satisfy what is about to be
                    -- committed, so update reservation
               SR(RemoteConnectionDescriptor)
          }
          if <(J.112 QoS) or (RSVP and ReserveDestination provided )> then {
               SC(RemoteConnectionDescriptor)
                    -- send to ReserveDestination if RSVP
          } else ERROR
```

```
     } else ERROR.  -- Cannot commit send direction without
                    -- RemoteConnectionDescriptor
}
ReceiveCommit()

If <current resources committed ≠ resources to commit> then {
                    -- skip commit if existing OK
          If not <resources to commit ⊂ resources reserved> then {
               If <RemoteConnectionDescriptor provided> then
                    RR(RemoteConnectionDescriptor)
          else if <(J.112 QoS) or (RSVP and ReserveDestination provided )> then
               RR(*)     -- Use wildcard classifier, send to
                         -- ReserveDestination if RSVP
          else ERROR
     }
          If <RemoteConnectionDescriptor provided> then
               RC(RemoteConnectionDescriptor)
          else if <(J.112 QoS) or (RSVP and ReserveDestination provided )> then
               RC(*)     -- Use wildcard classifier, send to
                         -- ReserveDestination if RSVP
          else ERROR
}
SendReserveDelete()

If <send resources reserved> then
          SD()      -- delete the reservation

ReceiveReserveDelete()

If <receive resources reserved> then
          RD()      -- delete the reservation

SendUnCommit()

If <send resources commit> then
          SU()      -- uncommit committed resources

ReceiveUnCommit()

If <receive resources committed> then
          RU()      -- uncommit committed resources

State UpdateState(DoCommit, DoReserve, OldState)

If <DoCommit = true> then
          return Commit
else if <DoReserve = true> then
          return Reserve
else
          return OldState
```

APPENDIX I

Example event package

This appendix provides an example event package for analogue access lines. The package is merely included here for illustrative purposes and to facilitate the inclusion of informative examples in the main part of the Recommendation. It does in no way constitute a complete package definition, nor should the package name shown be considered assigned. As the package is merely an example, details of individual events and signals are omitted here as well and only provided as high level descriptions for illustrative purposes.

Example line package

Package name: X

The following codes are used to identify events and signals for the "example line" package for "analogue access lines":

Code	Description	Event	Signal	Additional info
0-9,*,#,A, B,C,D	DTMF tones	√	BR	
bz	Busy tone	–	TO	
dl	Dial tone	–	TO	
hd	Off-hook transition	P, S	–	
hf	Flash hook	P	–	
hu	On-hook transition	P, S	–	
rg	Ringing	–	TO	
rt	Ring back tone	–	C, TO	
t	Timer	√	–	
vmwi	Visual Message Waiting Indicator	–	OO	
X	DTMF tones wildcard	√	–	Matches any of the digits "0-9"

As the above package is merely an example, the definition of the individual events and signals below is provided as a high-level description only. An actual and implementable package would have to specify the details of each event and signal. These details may differ between analogue PSTN service providers:

DTMF tones (0-9,*,#,A, B,C,D): Defines all of the DTMF tones.

Busy tone (bz): The busy tone indicates to the calling party that the called party is already engaged in a call.

Dial-tone (dl): The dial tone indicates to the calling party that a call can be placed.

Off-hook transition (hd): The off-hook event indicates that the phone associated with the endpoint went off-hook.

Flash hook (hf): The flash hook event indicates that a flash hook occurred on the phone associated with the endpoint.

On-hook transition (hu): The on-hook event indicates that the phone associated with the endpoint went on-hook.

Ringing (rg): The ringing signal indicates that the called party's telephone should be rung.

Ring back tone (rt): The ring back signal informs the calling party that the called party is being alerted.

Timer (t): As described in 6.1.5, timer T is a provisionable timer that can only be cancelled by DTMF input.

Visual Message Waiting Indicator (vmwi): The visual message waiting indicator signal either enables or disables a visual indication of a voice-mail message waiting.

DTMF tones wildcard (X): The DTMF tones wildcard matches any DTMF digit between 0 and 9.

APPENDIX II

Example command encodings

This appendix provides examples of commands and responses shown with the actual encoding used assuming the example line package is used. Examples are provided for each command. All commentary shown in the commands and responses is optional.

NotificationRequest

The first example illustrates a NotificationRequest that will ring a phone and look for an off-hook event:

```
RQNT 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0 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 empty[34], 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 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hd(A, E(S(dl), R(B/oc, hu, [0-9#*T](D))))
D: (0T|00T|#xxxxxxx|*xx|91xxxxxxxxxx|9011x.T)
S:
Q: process
T: ft
```

The response indicates that the transaction was successful:

```
200 1202 OK
```

Notify

The example below illustrates a Notify message that notifies an off-hook event followed by a 12-digit number beginning with "91". A transaction identifier correlating the Notify with the NotificationRequest it results from is included. The command is sent to the current "notified entity", which typically will be the actual value supplied in the NotifiedEntity parameter, i.e. "`ca@ca1.whatever.net:5678`" – a failover situation could have changed this:

```
NTFY 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0 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
```

---

[34] It could have been omitted as well.

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 μ-law will be the codec used and the packetization period will be 10 ms. The connection mode will be "receive only":

```
CRCX 1204 aaln/1@rgw-2567.whatever.net MGCP 1.0 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
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The second example illustrates a CreateConnection command containing a notification request and a RemoteConnectionDescriptor:

```
CRCX 1205 aaln/1@rgw-2569.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789AD
R: hd
S: rg
v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction failed, because the phone was already off-hook. Consequently, neither a connection-id nor a session description is returned:

```
401 1205 Phone off-hook
```

Our third example illustrates the use of the provisional response and the three-way handshake. We create another connection this time using dynamic quality of service and acknowledging the previous response received:

```
CRCX 1206 aaln/1@rgw-2569.whatever.net MGCP 1.0 NCS 1.0
K: 1205
C: A3C47F21456789F0
L: p:10, a:PCMU, dq-gi:A735C2
M: inactive

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
```

```
m=audio 3456 RTP/AVP 0
```

A provisional response is returned initially:

```
100 1206 Pending
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

A little later, the final response is received:

```
200 1206 OK
K:
DQ-RI: A12D5F1
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The Call Agent acknowledges the final response as requested:

```
000 1206
```

and the transaction is complete.

ModifyConnection

The first example shows a ModifyConnection command that simply sets the connection mode of a connection to "send/receive" – the "notified entity" is set as well:

```
MDCX 1209 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
N: ca@ca1.whatever.net
M: sendrecv
```

The response indicates that the transaction was successful:

```
200 1209 OK
```

In the second example, we pass a session description and include a notification request with the ModifyConnection command. The endpoint will start playing ring-back tones to the user:

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

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction was successful:

```
200 1206 OK
```

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 1.0 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
```

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 1.0 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
```

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 1.0 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 1.0 NCS 1.0
```

The response indicates success:

```
250 1210 OK
```

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" wildcard for the local portion of the endpoint-name:

```
AUEP 1200 *@rgw-2567.whatever.net MGCP 1.0 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 1.0 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:X;B, m:sendonly;
                    recvonly;sendrecv;inactive;netwloop;netwtest
A: a:G729A, p:30-90, e:on, s:on, v:X;B, 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 1.0 NCS 1.0
F: R,D,S,X,N,I,T,O,ES
```

The response indicates success:

```
200 2002 OK
R: X/hu,oc(N),[0-9](N)
D:
S: vmwi(+)
X: 0123456789B1
N: [128.96.41.12]
I: 32F345E2
T:
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 "X/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 empty, 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.

AuditConnection

The first example shows an AuditConnection command where we audit the CallId, NotifiedEntity, LocalConnectionOptions, Connection Mode, LocalConnectionDescriptor, and the Connection Parameters:

```
AUCX 2003 aaln/1@rgw-2567.whatever.net MGCP 1.0 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@ca1.whatever.net
```

```
L: p:10, a:PCMU
M: sendrecv
P: PS=395, OS=22850, PR=615, OR=30937, PL=7, JI=26, LA=47
v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0
```

In the second example, we request to audit RemoteConnectionDescriptor and LocalConnectionDescriptor:

```
AUCX 1203 aaln/2@rgw-2567.whatever.net MGCP 1.0 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
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0

v=0
```

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 1.0 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 1.0 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 6.4.3.5), this time going to Call Agent "`CA-1@whatever.net`".

APPENDIX III

Example call flow

In this appendix 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. Also, the call flow uses the example line package.

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	– – –	– – –	→	
		←	– – –	– – –	P-Ack(SDP2)	
		←	– – –	– – –	Ack(SDP2)	(ringing)
		Ack	– – –	– – –	→	
(ringback)	←	Modify Connection(SDP2) + Notification Request				
	Ack	→				
		←	– – –	– – –	Notify	Off-hook
		Ack	– – –	– – –	→	
	←	ModifyConnection + Notification Request				
	Ack	→				
	(cut in)	Call start	– – –	→		
		Notification Request	– – –	– – –	→	
		←	– – –	– – –	Ack	

Usr-1	EC-1	CA	CDB	ACC	EC-2	Usr-2
		(Call Established)				
		←	– – –	– – –	Notify	On hook
		Ack	– – –	– – –	→	
	←	Delete Connection				
		Delete Connection	– – –	– – –	→	
	Ack (Perf 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 1.0 NCS 1.0
N: ca@ca1.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 1.0 NCS 1.0
N: ca@ca1.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 digits[35]:

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

```
CRCX 1202 aaln/1@ec-1.whatever.net MGCP 1.0 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]xxxxxxxxx | 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
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The 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 μ-law transmission.

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

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. The Call Agent furthermore decides to acknowledge receipt of the responses for transaction 1202:

```
RQNT 1203 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
K: 1202
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 1.0 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789B0
R: hd
S: rg
```

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

The 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.

We assume this command does not finish executing immediately[36], and a provisional response is therefore returned by the egress embedded client acknowledging 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:

```
100 2001 Pending
I: 32F345E2

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1297 RTP/AVP 0
```

Once the transaction finishes execution, the embedded client sends the final response to the Call Agent, repeating the information it provided in the provisional response:

```
200 2001 OK
K:
I: 32F345E2

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1297 RTP/AVP 0
```

When the Call Agent receives the final response, it notices the presence of the empty Response Acknowledgement attribute and therefore issues a Response Acknowledgement for the transaction:

```
000 2001
```

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 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

---

[36] This could, e.g. be due to external resource reservation, although we did not include that in our example.

```
M: recvonly
X: 0123456789AE
R: hu
S: rt@FDE234C8

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
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 1.0 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 1.0 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 client[37]:

```
RQNT 2002 aaln/1@ec-2.whatever.net MGCP 1.0 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.

---

[37] 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 1.0 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 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8

DLCX 2004 aaln/1@ec-2.whatever.net MGCP 1.0 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 1.0 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 1.0 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 1.0 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 IV

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";
- "replicate";
- "conference".

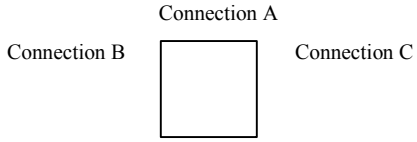
Connections in "network loopback", "network continuity test", or "inactive" modes are not affected by connections in the "active" modes. The table uses the following conventions:

- 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**	**replcate**
Connection B mode	**sendonly**	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$
	recvonly		$A_{out}=NA$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=NA$ $H_{out}=B_{in}$	$A_{out}=H_{in}+B_{in}$ $B_{out}=NA$ $H_{out}=B_{in}$
	sendrecv			$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=H_{in}+B_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$
	confrnce				$A_{out}=H_{in}+B_{in}$ $B_{out}=H_{in}+A_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=A_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$	$A_{out}=H_{in}+B_{in}$ $B_{out}=H_{in}$ $H_{out}=B_{in}$
	inactive					$A_{out}=NA$ $B_{out}=NA$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}=NA$ $H_{out}=NA$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=NA$
	netwloop/ netwtest						$A_{out}=A_{in}$ $B_{out}=B_{in}$ $H_{out}=NA$	$A_{out}=H_{in}$ $B_{out}=B_{in}$ $H_{out}=NA$
	replcate							$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=NA$

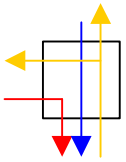
If there are three or more "active" channels they will still interact as defined in the table above with the outgoing media streams mixed for each interaction. (Union of all streams) If internal resources are used up and the streams cannot be mixed, the gateway should return a resources Not available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

These connections can be graphically represented as such:

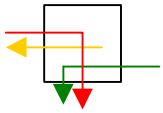


For example, if Connection A is Sendrecv, Connection B is confrnce, and Connection C is recvonly, from the above table the outputs in each mode will be:

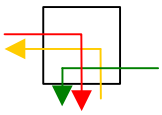
A to B Interaction: $B_{out}=H_{in}$ $A_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$



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



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



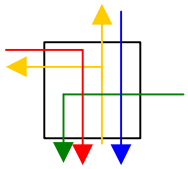
Taking the union of all streams in each output we get:

$A_{out}=H_{in}$

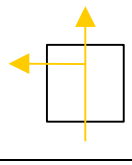
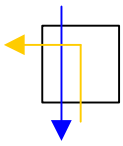
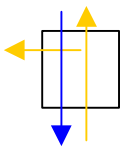
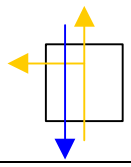
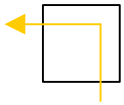
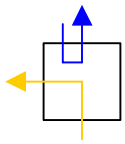
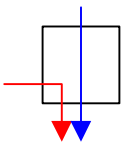
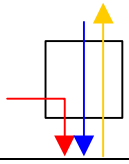
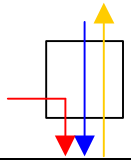
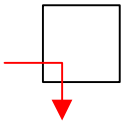
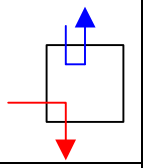
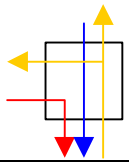
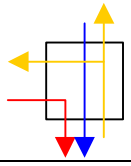
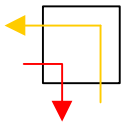
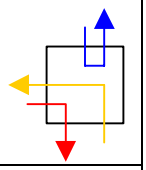
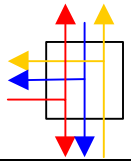
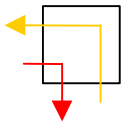
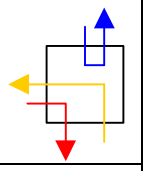
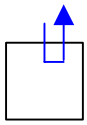
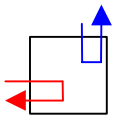
$B_{out}=H_{in}$

$C_{out}=NA$

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



For clarity, the table described above is repeated below in graphical form (excluding the "replicate" mode):

		Connection A mode (top)					
		sendonly	**recvonly**	**sendrecv**	**confrnce**	**inactive**	**netwloop/ netwtest**
Connection B mode (left)	**sendonly**						
	recvonly						
	sendrecv						
	confrnce						
	inactive						
	netwloop/ netwtest						

APPENDIX V

Compatibility information

This appendix provides NCS protocol compatibility information.

MGCP Compatibility

NCS is a profile of MGCP 1.0; however, NCS has introduced a couple of additions as well. The following lists NCS additions that are currently not included in MGCP:

- **Endpoint Naming Scheme** – The rules for wildcarding are more restrictive than in MGCP.
- **Embedded ModifyConnection** – A new Embedded ModifyConnection action has been introduced.
- **Dynamic Quality of Service** – IPCablecom Security services are supported in NCS. This affects the LocalConnectionOptions, Capabilities, and SDP. Also, a new return parameter; ResourceID, is added for CreateConnection and ModifyConnection.

- **Security** – IPCablecom Security services are supported in NCS. This affects the LocalConnectionOptions, Capabilities, and SDP.
- **Endpoint Name Retrieval –** The AuditEndpoint command has been extended with a capability to return the number of endpoints that match a wildcard as well as mechanism for block-wise retrieval of these endpoint names. Besides extending the AuditEndpoint command, this implies the introduction of two new parameter names: MaxEndPointIds, and NumEndPoints.
- **Supported Versions** – The RestartInProgress response and the AuditEndpoint command have been extended with a VersionSupported parameter to enable Call Agents and gateways to determine which protocol versions each support.
- **Error Codes** – Two new error codes have been introduced: 532 and 533.
- **Usage of SDP** – A new SDP usage profile is included in NCS. Most notably, the profile and all example use specifically require strict SDP compliance, regardless of the usefulness of the included fields. Also, IPCablecom specific extensions have been added to SDP.
- **Provisional Response** – Additional detail and specification of the provisional response mechanism has been included in NCS. A Response Acknowledgement response (000) has been introduced, an empty ResponseAck parameter has been permitted in final responses that follow provisional responses, and a procedure for the mechanism specified.
- **Signal Parameters** – Signal parameter syntax has been extended to allow for the usage of balanced parenthesis within signal parameters. All Time-Out signals can have their time-out value altered by a signal parameter.
- **Event Packages –** NCS introduces a set of new event packages.

Finally, it should be noted that NCS provides interpretations of and, in some cases, additional specification or clarification of the base MGCP protocol behaviour that may or may not reflect the intended MGCP behaviour.

APPENDIX VI

Additional example event packages

This appendix defines additional example event packages for the various types of endpoints currently defined for embedded clients.

Analogue Access Lines

The following package is currently defined for Analogue Access Line endpoints:

- Japanese Line;
- ADSI.

Japanese Line Package

Package name: J

The following codes are used to identify events and signals for the "Japanese line" package for "analogue access lines":

1) *Types of Subscriber Line Signalling*

   Subscriber line signals (signals) can be classified into signals related to connection control (Supervisory signal), those related to selection control (selection signals) and audible tone signals (audible tones).

2) *Supervisory Signals*

Code	Signal name	Event	Signal	Additional info
cs	Calling signal	P, S	–	Notification of originating call (=Off-hook transition)
ir	Ringing signal	–	TO	Notification of incoming call Time-out = infinite See Article 31, Item 2 in the Carriers Telecommunication Facilities Regulations
as1	Answer signal 1	P, S	–	Notification that called terminal has answered (Terminal to Network) (=Off-hook transition)
as2	Answer signal 2	–	TO	Notification that called terminal has answered (Network to Terminal) Time-out = infinite
ds1	Disconnect signal 1	P, S	–	Notification that communication is completed (Terminal to Network) (=On-hook transition)
ds2	Disconnect signal 2	–	TO	Notification that originating terminal has terminated communication (Network to Terminal) Time-out = infinite
cbs	Clear back signal	P, S	–	Notification that called terminal has terminated communication (=On-hook transition)
hs	Hooking signal	P	–	For "call waiting" and "three-party service"
sir	Extension call signal	–	TO	Outputted by the centralized extension system (CES) Time-out = infinite
tir	Callforward warning signal	–	TO	For "Voice Warp" service Time-out = 2-3 s
car	Data receiving terminal activation signal	–	TO	Notification by MODEM signal Time-out = infinite
pas	Primary answer signal	P, S	–	For Number Display. (=Off-hook transition)
iss	Incoming successful signal	P, S	–	For Number Display (=On-hook transition)
cei1(nu)	Callee ID(PB tone)	–	BR	"nu" denotes number
cei2(nu)	Callee ID(Modem tone)	–	BR	"nu" denotes number
ci	Caller ID	–	BR	"nu" denotes number
aw	Answer tone	✓	–	
ft	Fax tone	✓	–	
mt	Modem tone	✓	–	

Code	Signal name	Event	Signal	Additional info
ma	Media start	C	–	
oc	Operation complete	✓	–	
of	Operation failure	✓	–	
t	Timer	✓	–	
l	DTMF long duration	✓	–	
ld	Long duration connection	C	–	

3) *Selection Signal*

Code	Signal name	Event	Signal	Additional info
ssn	Selection Signal (0-9,*,#)	✓	BR	Partial Dial Time-out = 20-30 s Interdigit Time-out = 4-6 s
ssw	PB tones wildcard	✓	–	Matches any of the digits "0-9"

4) *Audible Tones*

Code	Signal name	Event	Signal	Additional info
dt	Dial tone	–	TO	Ready to receive selection signal Time out = 20-30 s
sdt	Second dial tone	–	TO	For register type services such as "call forwarding", "automatic telephone answering service" Time-out = 20-30 s
rbt	Ring back tone	–	C,TO	Time-out = infinite
bt	Busy tone	–	TO	Time-out = 60-70 s
cpt	Acceptance tone	–	BR	For register type services such as "call forwarding", "automatic telephone answering service"
hst	Hold service tone	–	TO	Time-out = infinite
iit	Incoming identification tone	–	C, BR	For "automatic telephone answering service"
siit	Specific incoming identification tone	–	C, BR	In case of double contract with "automatic telephone answering service" and "NARIWAKE service"
nft	Notification tone	–	TO	Only for "message identification reception service" Time-out = 3-4 s
how1	Howler tone 1	–	TO	Time-out = 10-22 s
how2	Howler tone 2	–	TO	Time-out = infinite

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

Calling signal (cs): Notifies the network of an originating call.

Ringing signal (ir): See Article 31, item 2 in the Carriers Telecommunication Facilities Regulations. The provisioning process may define the ringing cadence. The ringing signal may be parameterized with the signal parameter "rep" which specifies the maximum number of ringing cycles (repetitions) to apply. The following will apply the ringing signal for up to 6 ringing cycles:

```
S: ir(rep=6)
```

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.

Answer signal (as): Notifies the network that the called terminal has answered (as1). In reverse direction, the network notifies the original terminal that the called terminal has answered (as2).

Disconnect signal (ds): The originating terminal notifies the network that communication is completed (ds1). In reverse direction, the network notifies the called terminal that the originating terminal has terminated communication(ds2).

Clear back signal (cbs): Notifies the network that the called terminal has terminated communication.

Hooking signal (hs): The terminal notifies the network of an assignment or that a service has been changed during communication. This signal is used for "call waiting" and "three-party service".

Extension call signal (sir): With a Centralized Extension System (CES) telephone, the network notifies the terminal that an incoming call is being forwarded. In addition, for "NARIWAKE service", the network informs the terminal that there is an incoming call from a party that wants to be identified.

Call forward warning signal (tir): During the start of "Forwarding telephone" service or the unconditional transfer mode in "Voice warp", the network notifies the terminal that there is an incoming call to the subscribing customer and forwarding has been activated.

Data receiving terminal activation signal (car): The network notifies a data receiving terminal that there is a call incoming with information notified by modem signal.

Primary answer signal (pas): The called terminal notifies the network that the telephone set is hooked-off. This function is used for the number display.

Incoming successful signal (iss): The network notifies the originating terminal that the incoming signal is successfully received. This function is used for the number display.

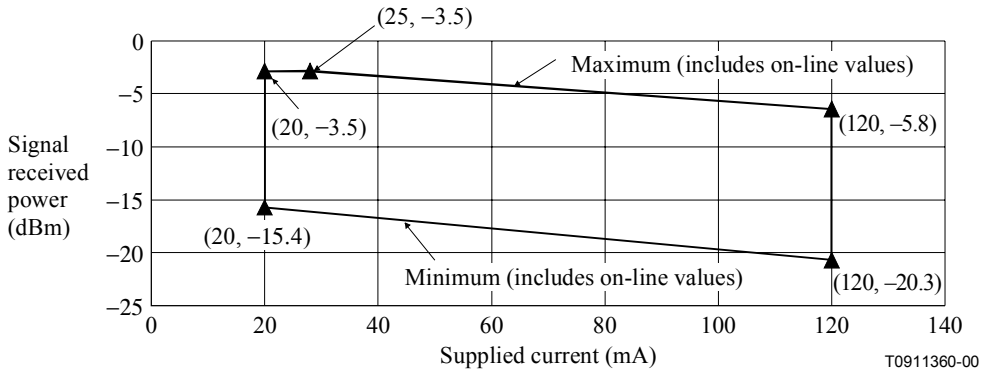
Selection signal (ss): The original terminal notifies the network of the type of service and the number of the other party. Code is assigned for Selection Signal (0-9, *, #) as ssn, PB tones wildcard as ssw. Frequencies and reception levels of PB (Push Button) dialing signals are shown in the following tables and figures.

1) *Frequency*

High group frequencies / **Low group frequencies**	**1209 Hz**	**1336 Hz**	**1477 Hz**
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

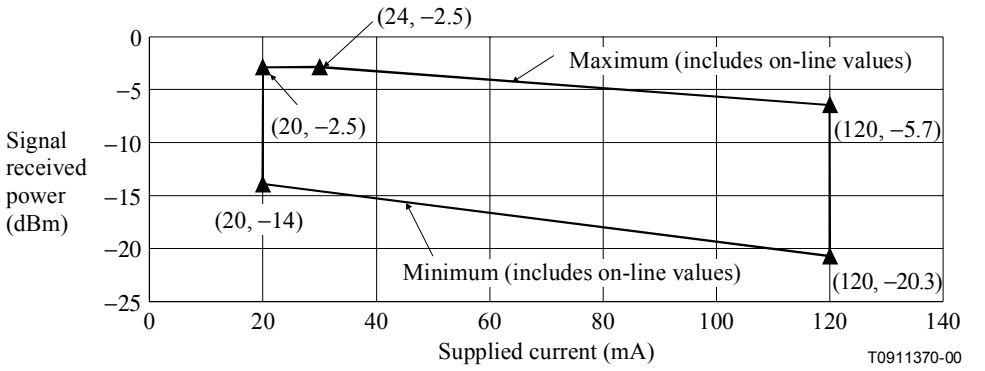
2) *Reception Standard*

Item		Standard
Signal frequency deviation		Within ±1.5%
Tolerance range of signal received power	Low group frequencies	Shown in Figure VI.1
	High group frequencies	Shown in Figure VI.2
	Electric power deviation between two frequencies	Within 5 dB, however the electric power for the low group frequency should be lower than that for the high group frequency.
Signal output time		50 ms or more
Minimum pause		30 ms or more
Cycle		120 ms or more
NOTE 1 – The minimum pause is the shortest dead time between adjacent signals. NOTE 2 – One cycle is the sum of signal send time and minimum pause.		



NOTE – The signal received power when the supplied current is less than 20 mA should be from −15.4 dBm to −3.5 dBm. When the supplied current is more than 120 mA, it should be from −20.3 dBm to −5.8 dBm.

Figure VI.1/J.162 – Tolerance range of signal received power (low group frequency)



NOTE – The signal received power where the supplied current is less than 20 mA should be from −14 dBm to −2.5 dBm. When the supplied current is more than 120 mA, it should be from −20.3 dBm to −5.7 dBm.

Figure VI.2/J.162 – Tolerance range of signal received power (high group frequency)

Other conditions are stipulated in Ordinance 13 of the Ministry of Posts and Telecommunications, 1998.

Dial tone (dt): The network notifies the originating terminal that it is ready to receive the selection signal. In off-net calling from a member network telephone, the network notifies the originating terminal that it is ready to receive the selection signal. Dial tone is an AC tone with frequency of 400 Hz and Levels between (–22–L) and −19 dBm where L is the transmission loss in a 400 Hz subscriber loop.

Second dial tone (sdt): The network notifies the originating terminal that it is ready to receive the second selection signal. In off-net calling from a member network telephone, the network notifies the originating terminal that it is ready to receive the selection signal. Second dial tone is an AC tone with frequency of 400 Hz and Levels between (–22–L) and −19 dBm where L is the transmission loss in a 400 Hz subscriber loop. The break-make ratio and make ratio are within 240 IPM and 50%, respectively.

Ringing back tone (rbt): The network notifies the originating terminal that it is calling the receiving terminal. The tone is terminated when an answer signal is received from the called terminal. Audible Ringback Tone is a combination of two AC tones with frequencies of 400 and 15-20 Hz and levels between −4 and (–29–L) dBm where L is the transmission loss in a 400 Hz subscriber loop. The break-make ratio and make ratio are within 20 IPM ± 20% and 33 ± 10%, respectively. (Modulation ration: within 85 ± 15%)

Busy tone (bt): The network notifies the originating terminal that the receiving terminal is in the communication status; thus, it cannot execute the service or connection that the originating terminal requested. Busy tone is an AC tone with frequency of 400 Hz and Levels between (–29–L) and −4 dBm where L is the transmission loss in a 400 Hz subscriber loop. The break-make ratio and make ratio are within 60 IPM ± 20% and 50 ± 10%, respectively.

Acceptance tone (cpt): The network notifies the originating terminal that it has received the service request. Acceptance tone is an AC tone with frequency of 400 Hz and Levels between (–26–L) and −16 dBm where L is the transmission loss in a 400 Hz subscriber loop.

Hold service tone (hst): The network notifies a waiting terminal that the wait state is continuing. Audible hold service tone is a combination of two AC tones with frequencies of 400 and 16 Hz and levels between −14 and (–22–L) dBm where L is the transmission loss in a 400 Hz subscriber loop. (Modulation ration: within 85%)

Incoming identification tone (iit): The network notifies the relevant called terminal that it has received an incoming call from a third party during conversation with a second party. Audible incoming identification tone is a combination of two AC tones with frequencies of 400 and 16 Hz and levels between −14 and (–25–L) dBm where L is the transmission loss in a 400 Hz subscriber loop. (Modulation ration: within 85%)

Specific incoming identification tone (siit): The network notifies the relevant called terminal that it has received an incoming call from a third party that has been identified. Audible specific incoming identification tone is a combination of two AC tones with frequencies of 400 and 16 Hz and levels between −14 and (–25–L) dBm where L is the transmission loss in a 400 Hz subscriber loop. (Modulation ration: within 85%)

Notification tone (nft): The network notifies the terminal of a customer subscribing to "message identification reception service" that it has received message identification. Notification tone is an AC tone with frequency of 400 Hz and Levels between (–26–L) and −16 dBm where L is the transmission loss in a 400 Hz subscriber loop.

Howler tone (how): The network notifies a terminal that an unused telephone receiver has been off-hook for a certain time to urge that the handset be placed on-hook. Two Howler tones are provided. Howler tone1(how1) is an AC tone with frequency of 400 Hz and levels +35 dBm or

under. Howler tone 1 is gradual increase sound for 3-15 seconds and a time-out signal for 10-22 seconds. Howler tone2(how2) is generated by combining three tones at frequencies of 1600 Hz, 1000 Hz and 2000 Hz at a cadence of 0.5 second of 1600 Hz, repeating twice of 0.125 second of 1000 Hz and 2000 Hz. The level of the combined tone is –1 dBm or less. Between these audible tones the voice guidance such as "The receiver is off-hook" is inserted. It is considered an error to try and play Howler tone on a phone that is on hook and an error should consequently be returned when such attempts are made. Howler tone2 has a time-out signal for infinite.

Callee ID (cei1(nu)): Direct inward dialing requires callee ID for PB signalling system.

Callee ID (cei2(nu)): Direct inward dialing requires callee ID for Modem signalling system.

Caller Id (ci(time, number, name)): 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 is coded as an ASCII character string of decimal digits that identify the calling line number. White spaces are permitted if the string is quoted; however, they will be ignored.
- The **name** parameter is coded as a string of ASCII characters that identify the calling line name. White spaces are permitted if the string is quoted.

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. The following example illustrates the use of the caller-id signal:

```
S: ci(02/20/19/47, "5273 4671", JCTEA)
```

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 V.8.

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

Media start (ma): The media start event occurs on a connection when the first valid[38] RTP media packet is received on the connection. This event can be used to synchronize a local signal, e.g. ringback, with the arrival of media from the other party.

The event may be detected on a connection. When no connection is specified, the event applies to all connections for the endpoint, regardless of when the connections are created.

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

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 dialled 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/dt)
```

[38] When authentication and integrity security services are used, an RTP packet is not considered valid until it has passed the security checks.

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: L/oc(L/rbt@0A3F58)
```

When the operation complete event is requested, it cannot be parameterized with any event parameters. When the package name is omitted, the default package name is assumed.

The operation complete event may additionally be generated as defined in the base protocol, e.g. when an embedded Modify Connection command completes successfully, as in[39]:

```
O: L/oc(B/C)
```

Operation failure (of): In general, the operation failure event may be generated when the endpoint was asked to apply one or several signals of type TO on the endpoint, and one or more of those signals failed prior to timing out. The completion report may carry as a parameter the name of the signal that failed, as in:

```
O: L/of(L/ir)
```

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: L/of(L/rbt@0A3F58)
```

When the operation failure event is requested, event parameters can not be specified. When the package name is omitted, the default package name is assumed.

The operation failure event may additionally be generated as specified in the base protocol, e.g. when an embedded Modify Connection command fails, as in[39]:

```
O: L/of(B/C(M(sendrecv(AB2354))))
```

Timer (t): 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:   dt
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)
```

Note that only one of the two forms can be used at a time, since a given event can only be specified once.

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.

---

[39] Note the use of "B" here as the prefix for the parameter reported.

DTMF Long duration (l): The "DTMF Long duration" is observed when a DTMF signal is produced for a duration longer than 2 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.

The event may be detected on a connection. When no connection is specified, the event applies to all connections for the endpoint, regardless of when the connections are created.

PB tones wildcard (x): The PB tones wildcard matches any PB digit between 0 and 9.

ADSI Package

Package name: JS

Code	Signal name	Event	Signal	Additional info
adsi(string)	ADSI display	–	BR	

ADSI display (adsi(string)): Analogue Display Services Interface (ADSI) is mainly used for display of the originator's telephone number. See 4.2, Receiving Functions for the Originator's Telephone Number (Number Display), Technical Reference of Telephone Service Interfaces.

Video

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

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2002