International Telecommunication Union

ITU-T J.171.1

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(11/2005)

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

IPCablecom

IPCablecom trunking gateway control protocol (TGCP): Profile 1

ITU-T Recommendation J.171.1

ITU-T Recommendation J.171.1

IPCablecom trunking gateway control protocol (TGCP): Profile 1

Summary

This Recommendation describes an IPCablecom profile of an Application Programming Interface (API) called a Media Gateway Control Interface (MGCI) and a corresponding protocol (MGCP) for controlling voice-over-IP (VoIP) PSTN gateways from external call control elements. This is one of the two profiles referenced in ITU-T Rec. J.171.0. The second profile is specified in ITU-T Rec. J.171.2.

The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements. The IPCablecom profile as described in this Recommendation will be referred to as the IPCablecom Trunking Gateway Control Protocol (TGCP).

Source

ITU-T Recommendation J.171.1 was approved on 29 November 2005 by ITU-T Study Group 9 (2005-2008) under the ITU-T Recommendation A.8 procedure.

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.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

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.171.1

IPCablecom trunking gateway control protocol (TGCP): Profile 1

1 Scope

This Recommendation describes an IPCablecom profile of an Application Programming Interface (API) called a Media Gateway Control Interface (MGCI) and a corresponding protocol (MGCP) for controlling voice-over-IP (VoIP) PSTN gateways from external call control elements. This is one of the two profiles referenced in ITU-T Rec. J.171.0. The second profile is specified in ITU-T Rec. J.171.2.

The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements. The IPCablecom profile as described in this Recommendation will be referred to as the IPCablecom Trunking Gateway Control Protocol (TGCP).

This Recommendation is based on the IPCablecom network-based call signalling Recommendation (ITU-T Rec. J.162), and IETF RFC 2705, *Media Gateway Control Protocol (MGCP)*. This Recommendation, which defines the IPCablecom TGCP protocol, constitutes a specification that is independent of MGCP. The TGCP profile of MGCP is strictly and solely defined by the contents of this Recommendation.

2 References

2.1 Normative references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

– ITU-T Recommendation J.161 (Draft, version 2), *Audio codec requirements for the provision of bidirectional audio service over cable television networks using cable modems*.

– ITU-T Recommendation J.162 (2005), *Network call signalling protocol for the delivery of time-critical services over cable television networks using cable modems*.

– ITU-T Recommendation J.170 (2005), *IPCablecom security specification*.

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

2.2 Informative references

– ITU-T Recommendation E.180/Q.35 (1998), *Technical characteristics of tones for the telephone service*.

– ITU-T Recommendation J.163 (2005), *Dynamic quality of service for the provision of real-time services over cable television networks using cable modems*.

– ITU-T Recommendation J.171.0 (2005), *IPCablecom trunking gateway control protocol (TGCP): Profiles overview*.

– IETF RFC 1889 (1996) *RTP: A Transport Protocol for Real-Time Applications*.

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

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

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

– TCP/IP Illustrated, Volume 1 (2001), *The Protocols,* Addison-Wesley, 1994.

3 Definitions

This Recommendation defines the following terms:

3.1 cable modem: The delivery of high-speed data access to customer locations using equipment built in conformance with ITU-T Recs J.83 and J.112.

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

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

DNS	Domain Name System
IP	Internet Protocol
IPSec	Internet Protocol Security
ISUP	ISDN User Part
MGC	Media Gateway Controller
MGCP	Media Gateway Control Protocol
MIB	Management Information Base
MTA	Media Terminal Adapter
MWD	Maximum Waiting Delay
NCS	Network-based Call Signalling
NTP	Network Time Protocol
QoS	Quality of Service
RTCP	Real-Time Control Protocol
RTO	Retransmission Timeout
RTP	Real-Time Protocol
SDP	Session Description Protocol
SG	Signalling Gateway
SPI	Security Parameters Index

5 Convention

If this Recommendation is implemented, the keywords "MUST" and "SHALL" as well as "REQUIRED" are to be interpreted as indicating a mandatory aspect of this Recommendation.

The key words indicating a certain level of significance of particular requirements that are used throughout this Recommendation are summarized as below.

"MUST" This word or the adjective "REQUIRED" means that the item is an absolute requirement of this Recommendation.

"MUST NOT" This phrase means that the item is an absolute prohibition of this Recommendation.

"SHOULD" This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

"SHOULD NOT" This phrase means that there may exist valid reasons in particular circumstances when the listed behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.

"MAY" This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

5.1 Background

This Recommendation describes an IPCablecom profile of an application programming interface called a Media Gateway Control Interface (MGCI) and a corresponding protocol (MGCP) for controlling voice-over-IP (VoIP) PSTN Gateways from external call control elements. The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements. The IPCablecom profile as described in this Recommendation will be referred to as the IPCablecom Trunking Gateway Control Protocol (TGCP) Profile 1.

This Recommendation is based on the IPCablecom Network-based Call Signaling (NCS) ITU-T Rec. J.162, the Media Gateway Control Protocol (MGCP) 1.0 IETF RFC 3435 (which was the result of a merge of the IETF draft of Simple Gateway Control Protocol and the IETF draft of IP Device Control (IPDC) family of protocols), and input generated by the IPCablecom PSTN Gateway focus team.

This Recommendation, which defines the IPCablecom TGCP Protocol, constitutes a document that is independent of MGCP in order to provide a stable reference document while meeting current time-to-market demands for such a reference. It is the intent of this Recommendation to be as closely aligned with NCS and MGCP 1.0 as possible for the IPCablecom environment, in order to avoid developing multiple protocols to solve the same problem. This goal has been and continues to be pursued through cooperation with the authors of the NCS and MGCP Recommendations. The TGCP profile of MGCP, however, is strictly and solely defined by the contents of this Recommendation.

This TGCP profile of MGCP, which will be referred to as the PSTN Trunking Gateway Call Signaling Protocol 1.0, TGCP 1.0, the TGCP profile, or simply TGCP in this Recommendation, has been modified from the MGCP 1.0 IETF RFC 2435 in the following ways:

- *The TGCP protocol only aims at supporting IPCablecom voice-over-IP PSTN Gateways.* The TGCP protocol supports voice-over-IP PSTN Gateways as defined by IPCablecom. Functionality present in the MGCP 1.0 protocol, which was superfluous to TGCP, has been removed.

- *The TGCP protocol contains extensions and modifications to MGCP*. IPCablecom-specific requirements are addressed in TGCP. However, the MGCP architecture, and all of the MGCP constructs relevant to PSTN Gateways, have been preserved in TGCP.
- *The TGCP protocol contains minor simplifications from MGCP 1.0.* Where several choices were available, and not necessarily needed for a PSTN Gateway in the IPCablecom environment, some simplifications have been made for trunking gateway implementations.

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

TGCP is designed to meet the protocol requirements for the Media Gateway Controller to Media Gateway interface defined in the IPCablecom architecture.

6 Introduction

This Recommendation describes the TGCP profile of an application programming interface (MGCI) and a corresponding protocol (MGCP) for controlling trunking gateways from external call control elements. A trunking gateway is a network element that provides analog, emulated analog, or digital bearer and channel-associated signalling trunk circuit access to a voice-over-IP (VoIP) network.

Trunking gateways are used for interfacing to the PSTN and as such are expected to adhere to the relevant electrical, operational, and signalling standards for the trunk type they implement. The TGCP Recommendation is developed for CLECs, and in the first version of this Recommendation the trunk types that will be supported are limited to the following:

- SS7 ISUP:
  - Standard Bearer trunk.
- MF trunks on a combined Equal Access End Office/Access Tandem (EAEO/AT):
  - Operator Services trunks[1];
  - CLEC subscriber access through a LEC Access Tandem;
  - Operator access to EAEO/AT for Busy Line Verification and Barge In;
  - Emergency service trunks for access to an emergency service Tandem.

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 Media Gateway Controllers. The MGCP assumes that these call-control elements, or Media Gateway Controllers (MGCs), 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 Media Gateway Controllers, although future IPCablecom Recommendations 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 a trunk circuit on a trunking gateway that terminates an ISUP bearer trunk to a switch. Another example is an embedded client or residential gateway that terminates residential POTS lines (to phones), although such devices are not addressed by this Recommendation.

---

[1] Operator Services are expected to be provided by or accessed through a LEC.

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 TGCP 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.

Media Gateway Controllers instruct the gateways to create connections between endpoints and to detect certain events, e.g., continuity test, and generate certain signals, e.g., ringback. It is strictly up to the Media Gateway Controller 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 Media Gateway Controller 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 Media Gateway Controllers 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 Media Gateway Controller's instructions as they are received.

6.1 Relation with other IPCablecom standards

An IPCablecom PSTN Gateway consists of three functional components:

- Media Gateway Controller (MGC), which contains the call intelligence and terminates call signalling. This component is also referred to as a Call Agent.
- Media Gateway (MG), which terminates the bearer channels under the instruction and control of the MGC. This function is also known as a Trunking Gateway (TGW);
- Signalling Gateway (SG), which connects the call signaling to the PSTN and provides a signalling translation function.

In addition to the PSTN Gateway protocols, IPCablecom also includes a Network-based Call Signaling (NCS) protocol, which is a profile of MGCP. The SDP standard has a pivotal status in this architecture. Both NCS and TGCP use the Session Description Protocol (SDP) to carry session descriptions.

IPCablecom NCS and TGCP systems hold all of the call state in CMS and MGC Call Agents. Figure 1 below illustrates the relationships among these different components:

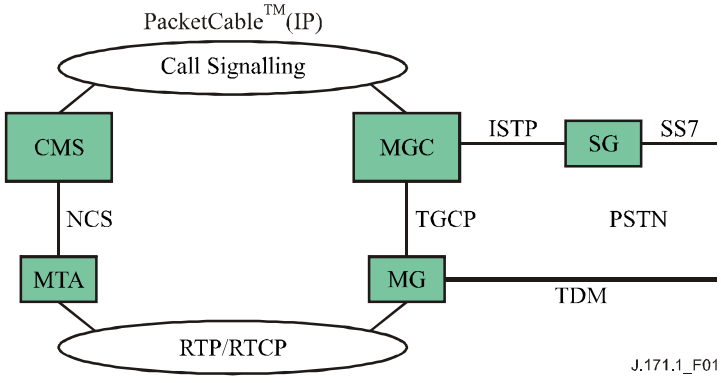


Figure 1/J.171.1 – Relationship among NCS and TGCP components

6.2 Relation to RFC 3435 and ABNF grammar

RFC 3435 includes a formal description of the MGCP protocol syntax following the "Augmented BNF for Syntax Specifications". This formal description is referenced by developers for the creation of interoperable devices. A copy of the MGCP protocol syntax, annotated and edited to indicate its applicability to IPCablecom Recommendations, is provided in Appendix VI. Adherence to these guidelines can improve interoperability by minimizing failures caused by different interpretations of syntax and grammar.

7 Media Gateway Control 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.

7.1 Model and naming conventions

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 MGCs. It should nonetheless be recognized that in none of these cases is a "connection" established within an IPCablecom network, as the term "connection" is understood within the circuit-switched PSTN. The terms "call" and "connection" in this context (and throughout this Recommendation) are used for convenience of reference, not to indicate any actual technical or other similarity between the IPCablecom network and the PSTN.

7.1.1 Endpoint names

Endpoint names, also known as, 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 IETF RFC 1034 and includes a host portion; thus, an example domain-name could be:

```
MyTrunkingGateway.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 IETF RFC 821 for details. However, use of IP addresses is generally discouraged.

Trunking gateways have one or more endpoints (e.g., one for each trunk) 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, e.g., DS-0, or an analog access line. The 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 wild-carding ("*", "$"), and white space characters.

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

  `term1/term2/term3`

  and one of the terms of the local endpoint name is wild-carded, then the local endpoint name looks like this:

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

  In each of the examples, a dollar sign could have appeared instead of the asterisk.
- Wild-carding is only allowed from the right: thus if a term is wild-carded, then all terms to the right of that term must be wild-carded as well.
- In cases where mixed dollar sign and asterisk wild cards are used, dollar-signs are only allowed from the right: thus if a term had a dollar sign wild card, all terms to the right of that term must also contain dollar sign wild cards.
- A term represented by an asterisk is to be interpreted as: "use *all* values of this term known within the scope of the trunking gateway in question". Unless specified otherwise, this refers to all endpoints configured for service, regardless of their actual service state, i.e., in-service or out-of-service.
- 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 trunking gateway in question". Unless specified otherwise, this only refers to endpoints that are in-service.
- 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. Note that since the domain name portion is part of the endpoint identifier, different forms or different values referring to the same entity are not freely interchangeable. Following a restart the most recently supplied form and value MUST always be used.

7.1.1.1 Trunking gateway endpoint names

Endpoints in trunking gateways MUST support the additional naming conventions specified in this clause.

Trunking gateways will support the following basic endpoint-type:

- `ds` A DS-0 trunk.

  The basic endpoint type is expected to be provisioned with additional information about the type of signalling supported on the trunk circuit and the switching system role it provides.

7.1.1.1.1 Trunk circuit endpoints

In addition to the naming conventions specified above, local endpoint names for PSTN trunking gateway endpoints of type "`ds`" MUST adhere to the following:

- Local endpoint names will consist of a series of terms each separated by a slash ("/") that describe the physical hierarchy within the gateway:

  `ds/<unit-type1>-<unit #>/<unit-type2>-<unit #>/.../<channel #>`
- The first term (`ds`) identifies the endpoint naming scheme used and the basic endpoint type.

- The last term is a decimal number that indicates the *channel* number[2] at the lowest level of the hierarchy.
- Intermediate terms between the first term (`ds`) and last term (channel number) represent intermediate levels of the hierarchy and consist of <unit-type> and <unit #> separated by a hyphen ("-") where:
  - the <unit-type> identifies the particular hierarchy level. Values of <unit-type> presently defined are: "s", "su", "oc3", "ds3", "e3", "ds2", "e2", "ds1", "e1" where "s" indicates a slot number and "su" indicates a sub-unit within a slot. Other values representing physical hierarchy levels that have not been included in this list but which follow the same basic naming rules will also be allowed;
  - the <unit #> is a decimal number which is used to reference to a particular instance of a <unit-type> at that level of the hierarchy.
- The number of levels and naming of those levels is based on the physical hierarchy within the media gateway, as illustrated by the following examples:
  - A Media Gateway that has some number of DS1 interfaces:

    `ds/ds1-#/#`
  - A Media Gateway that has some number of OC3 interfaces, that contain channelized DS3 and DS1 hierarchies:

    `ds/oc3-#/ds3-#/ds1-#/#`
  - A Media Gateway that contains some number of slots with each slot having some number of DS3 interfaces:

    `ds/s-#/ds3-#/ds1-#/#`
- Some endpoints may not contain all possible levels of a hierarchy; however, all levels supported by a given endpoint are contained in the endpoint naming scheme. For example, a DS3 without DS1 framing could be represented by the following naming scheme:

  `ds/s-#/ds3-#/#`

  However, a DS3 *with* DS1 framing could not be represented by that naming scheme.
- Wild-card naming follows the conventions stated in 7.1.1 with the asterisk character ("*") referring to "all", and the dollar character ("$") referring to "any". A range "[N-M]" wild-carding convention representing a "range" of channels from channel N to channel M inclusive is supported as well:
  - it should be noted that the use of the "all" wild-card for the first (`ds`) term refers to all endpoint-types in the media gateway regardless of their type. Use of this feature is generally expected to be for administrative purposes, e.g., auditing or restart;
  - a local endpoint name may be under-specified by supplying some smaller than normal number of terms starting from the left-hand side of the endpoint name. In that case, missing terms to the right of the last term specified are assumed to be the wild-card character "*", referring to "all of", unless the terms specified contain the "any of" wild-card character, in which case missing terms to the right of the last term specified are assumed to be the "any of" wild-card character;
  - wherever use of the "all" wild-card is permitted, the range of channels "[N-M]" wild-card may be used in the last term (i.e. <channel-#>) of the local endpoint name instead. The "range" wild-card will then refer to all of the channels from N to M. The rules and

[2] Please note the use of the term "channel" as opposed to "time slot".

restrictions that apply to the use of the "all" wild-card also apply to the use of the "range" wild-card.

The following examples illustrate the use of wild-carding:

`ds/ds1-3/*`	All channels on ds1 number 3 on the media gateway in question.
`ds/ds1-3/$`	Any channel on ds1 number 3 on the media gateway in question.
`ds/*`	All trunk-circuit endpoints on the media gateway in question.
`*`	All endpoints (regardless of endpoint-type) on the media gateway in question.
`ds/ds1-3/[1-24]`	Channels 1 to 24 on ds1 number 3 on the media gateway in question.

The above define the canonical names for endpoints in a trunking gateway. It is expected that aliasing may be supported in a future version of this Recommendation, e.g., to support bonding of multiple DS-0 trunks for video calls, e.g., on the form "`ds/ds1-1/H0-1`".

7.1.2 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 MGC. Call identifiers with a maximum length of 32 characters MUST be supported.

At a minimum, call identifiers MUST be unique within the collection of MGCs that control the same gateways. However, the coordination of these call identifiers between MGC's is outside the scope of this Recommendation. When an MGC builds several connections that pertain to the same call, either on the same gateway or in different gateways, these connections will all 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.

7.1.3 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.

7.1.4 Names of media gateway controllers and other entities

The Media Gateway Control Protocol has been designed for enhanced network reliability to allow implementation of redundant MGCs. This means that there is no fixed binding between entities and hardware platforms or network interfaces.

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

```
mgc1@mgc.whatever.net
```

Reliability is provided by the following precautions:

- Entities such as trunking gateways or MGCs 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 System (DNS). MGCs 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 failover in MGCP. The "notified entity" for an endpoint is the MGC 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 MGC, it MUST send the command to the current "notified entity" for which endpoint(s) the command pertains. Upon start up, the "notified entity" MUST be set to a provisioned value. Most commands sent by the MGC 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[3], 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 7.4 contains a more detailed description of reliability and failover.

7.1.5 Digit maps

In MGCP, the MGC can ask the gateway to collect digits dialled by a user. This facility is typically used by analog access lines with residential gateways to collect the numbers that a user dials, or it can be used for CAS PBX interfaces. Rather than sending each digit to the MGC as the digits are detected, the MGC can provide a grammar describing how many digits should be accumulated before the MGC is notified. This grammar is known as a *digit map*.

None of the trunk types supported by the current version of the TGCP Recommendation have a need for digit maps, and digit maps are therefore not part of this Recommendation.

7.1.6 Events and signals

The concept of events and signals is central to MGCP. An MGC may ask to be notified about certain events occurring in an endpoint, e.g., off-hook events. An MGC also may request certain signals to be applied to an endpoint, e.g., ringback.

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 ISUP trunks, and another package may support another group of events and signals for MF trunks. 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 ISUP trunk circuit with the ISUP package (package name "IT") being the default package, the following two event names are considered equal:

`IT/oc` Operation Complete in the ISUP package for an ISUP trunk circuit.

`oc` Operation Complete in the ISUP package (default) for an ISUP trunk circuit.

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

Table 1 below lists trunking gateway endpoint-types and the packages defined for them in this Recommendation. The MGC MUST support all packages listed in Table 1. The MG MUST support the "IT", "FXR", and "XRM" packages. The MG SHOULD support the "MO" and "MT" packages.

Table 1/J.171.1 – Packages Associated with endpoint-types

Endpoint-type	Package	Package name	Default package	1st release that introduced package support requirement
DS-0	ISUP trunk	IT	Yes	1.0
DS-0	MF OSS	MO	No	1.5
DS-0	MF Terminating	MT	No	1.5
DS-0	FAX	FXR	No	1.5
DS-0	VoIP Metrics	XRM	No	1.5

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 TGCP 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 ringback or fax tones. Events must further specify if they are persistent (see 7.3.1) and if they contain auditable event-states (see 7.3.8.1). 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 7.3.1.

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. A gateway 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 wild-card notation each. The wild-card character "*" (asterisk) can be used to refer to all packages supported by the endpoint in question, and the event code "all" to all events in the package in question. For example:

`IT/all` refers to all events in the ISUP trunk package for an ISUP trunk circuit.

`*/all` for an ISUP trunk circuit; 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:

```
IT/rt@0A3F58
```

Should the connection be deleted while an event or signal is being detected or applied on it, that particular event detection or signal generation MUST stop. Depending on the signal type the endpoint SHOULD generate a failure, i.e., if the signal type is TO, the "operation failure" event will be generated since the connection associated with the signal was deleted prior to the signal timing out. Notification action associated with reporting the failure must conform to the notify operations as defined for NotificationRequest handling (clause 7.3.1).

The wild-card 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:

```
IT/ma@*
```

However, when the event is actually observed, the gateway MUST include the name of the specific connection on which the event occurred. The wild-card 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:

```
IT/rt@$
```

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

The connection id, or a wild-card 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). However, the use of the "all packages" and "all events" wildcards are strongly discouraged. Call Agents must be able to operate in an environment where some endpoints do not support all packages. An endpoint that receives a command referencing a package it does not support will respond with error 518 (Unsupported or Unknown Package). On receiving this error response, the Call Agent could try the command again without the package parameter, although in the case where the original command contained parameters for multiple packages, the Call Agent would have no way of knowing which specific package(s) to exclude. The Call Agent can also use the AuditEndpoint command to determine the set of packages supported by an endpoint.

7.2 SDP use

The MGC 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 documented in IETF RFC 2327. In addition, all MGCs and media gateways MUST ignore any SDP parameters, attributes, or fields that are not understood by the call agent or gateway.

SDP allows for description of multimedia conferences. The TGCP profile will only support the setting of audio connections using the media type "audio".

SDP allows for description of real-time fax using media type "image". The TGCP profile will support setting of fax connections using media type "image".

7.3 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 control interface (MGCI). An MGCI function is defined for each MGCP command, where the MGCI function takes and returns the same parameters as the corresponding MGCP command. The functions shown in this 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 this 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 clause 8. Trunking gateways and Call Agents 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 MGC can issue a NotificationRequest command to a gateway, instructing the gateway to watch for specific events such as seizure or fax tones on a specified endpoint.
- The gateway will then use the Notify command to inform the MGC when the requested events occur on the specified endpoint.
- The MGC can use the CreateConnection command to create a connection that terminates in an endpoint inside the gateway.
- The MGC can use the ModifyConnection command to change the parameters associated to a previously established connection.
- The MGC 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 MGC 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 trunking gateway and each of the trunk circuits. 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 MGC 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 MGC) 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, a trunking gateway endpoint is limited to a specific trunk circuit within a trunking gateway.

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), "inactive" (inactive), "loopback" (loopback), "continuity test" (conttest), "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 MUST be sent on any connection whose mode is either "send only", or "send/receive ", unless the endpoint has a connection in "loopback" or "continuity test" mode. However, audio generated by applying a signal to a connection MUST be sent on the connection for all modes except "network loopback".

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", "loopback" or "continuity test" mode MUST be discarded.
- Audio signals received in data packets through connections in "receive only", or "send/receive" mode MUST be mixed and sent to the endpoint[4], unless the endpoint has another connection in "loopback" or "continuity test" mode.
- Audio signals originating from the endpoint MUST be transmitted over all the connections whose mode is "send only", or "send/receive" unless the endpoint has another connection in "loopback" or "continuity test" mode.

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

The "loopback" and "continuity test" modes are used during maintenance and continuity test operations. An endpoint may have more than one connection in either "loopback" or "continuity test" mode. As long as there is one connection in that particular mode, and no other connection on the endpoint is placed in a different maintenance or test mode, the maintenance or test operation MUST continue undisturbed. There are two variations of continuity test (COT), one specified by ITU, and one used in several national networks. In the first case, the test is a loopback test. The originating switch will send a tone (the go tone) on the bearer circuit and expect the terminating switch to loopback the circuit. If the originating switch sees the same tone returned (the return tone), the COT has passed. If not, the COT has failed. In the second case, the go and return tones are different. The originating switch sends a certain go tone. The terminating switch detects the go tone, it asserts a different return tone in the backwards direction. When the originating switch detects the return tone, the COT is passed. If the originating switch does not detect the return tone within a certain period of time, the COT has failed.

If the mode is set to "loopback", the gateway MUST return the incoming signal from the endpoint back into that same endpoint. This procedure will be used, typically, for testing the continuity of trunk circuits according to ITU specifications. If the mode is set to "continuity test", the gateway is informed that the other end of the circuit has initiated a continuity test procedure according to procedures specified for several national networks. The gateway will place the circuit in the transponder mode required for dual-tone continuity tests.

Furthermore, when a connection for an endpoint is in "loopback" or "continuity test" mode:

- audio signals received on any connection for the endpoint MUST *not* be sent to the endpoint;
- audio signals received on the endpoint MUST *not* be sent to any connection for the endpoint.

If the mode is set to "network loopback", the audio signals received from the connection MUST be echoed back on the same connection. The "network loopback" mode SHOULD simply operate as an RTP packet reflector. The media MUST not be forwarded to the endpoint.

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

4 TGCP endpoints are currently not required to support mixing however.

equipment to verify proper operation. The signal MUST go through internal decoding and re-encoding prior to being passed back. For DS-0 endpoints the signal will be an audio signal, and the signal MUST NOT be passed on to a circuit connected to the endpoint, regardless of the current seizure-state of that circuit.

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

Please refer to Appendix I for illustrations of mode interactions.

7.3.1 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 MGC, may then decide that a different type of encoding should be used on the connections bound to this endpoint and instruct the gateway accordingly[5].

```
ReturnCode
     ← NotificationRequest(EndpointId
                         [, NotifiedEntity]
                         [, RequestedEvents]
                         , RequestIdentifier
                         [, 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 7.1.1. The "any of" wild-card MUST NOT be used. A gateway that receives a NotificationRequest with the "any of" wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response. The "all of" wildcard MUST be supported for NotificationRequests with each of RequestedEvents, SignalsRequest, and DetectEvents being either empty or omitted. For simplicity, some gateways may choose to not support the "all-of" wildcard for NotificationRequests where one or more of these parameters is neither empty nor omitted. Such gateways shall respond with error code 503 if they receive an "all-of" wildcarded NotificationRequest which they are unable to process for this reason.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint. When used, the entire Media Gateway Controller name MUST be specified which includes both the local name and domain name – even if a bracketed IP address is used for the domain name. See 7.1.1 and 7.1.4 for more information. If, however, only the domain name is provided, the MG SHOULD use the domain name as the Call Agent ID.

RequestIdentifier is used to correlate this request with the notification it may trigger. It will be repeated in the corresponding Notify command.

SignalRequests is a parameter that contains the set of signals that the gateway is asked to apply. Unless otherwise specified, signals are applied to the endpoint; however, some signals can be applied to a connection. The following are examples of signals[6]:

- Continuity test;
- Set up MF OSS call.

---

5 The new instruction would be a ModifyConnection command.

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

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. Once turned on, it MUST NOT be turned off until explicitly instructed to by the MGC, 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 A.1 for further definition of this event). A TO signal could be "place MF call" timing out after 16 seconds. If an event occurs prior to the 16 seconds, the signal will, by default, be stopped[7]. If the signal is not stopped, the signal will time out, stop and generate an "operation complete" event, about which the MGC may or may not have requested to be notified. If the MGC has asked for the "operation complete" event to be notified, the "operation complete" event sent to the MGC will include the name(s) of the signal(s) that timed out[8]. 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 time out[8].
- `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.

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 tone is applied to an endpoint involved in an active communication, only the party using the endpoint in question will hear the tone. However, individual signals may define a different behaviour.

When a signal is applied to a connection that has received a RemoteConnectionDescriptor (see 7.3.3), 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 omission of the SignalRequests parameter is interpreted as an empty SignalRequests list.

---

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

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

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:

- seizure;
- fax tones;
- operation complete;
- incoming MF call.

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;
- 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[9]. 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[10]. 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 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. A given event MUST NOT appear more than once in a RequestedEvents. The omission of the RequestedEvents parameter is interpreted as an empty RequestedEvents list. More than one action can be specified for an event, although a given action cannot appear more than once for a given event. The matrix in Table 2 specifies the legal combinations of actions:

---

9 Thus, the RequestIdentifier will be the RequestIdentifier of the current NotificationRequest.

10 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.

Table 2/J.171.1 – Actions associated with events

Event	Notify	Accumulate	Ignore	Keep signal(s) active	Embedded Notification Request	Embedded ModifyConnection
Notify	–	–	–	√	–	√
Accumulate	–	–	–	√	√	√
Ignore	–	–	–	√	–	√
Keep signal(s) active	√	√	√	–	√	√
Embedded NotificationRequest	–	√	–	√	–	√
Embedded ModifyConnection	√	√	√	√	√	–

If a client receives a request with an invalid action or illegal combination of actions, it MUST return an error to the MGC (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 MGC can send a NotificationRequest with an empty RequestedEvents list to the gateway. However, persistent events will still be detected and notified.

When a stimulus that triggers multiple requested events is detected (e.g., fax tone is the stimulus for both FXR/gwfax(start), and L/ft), the gateway MUST generate only one of the events (namely the most preferred event out of the multiple requested events that triggered) based on the following precedence rules:

1) Events included in a RequestedEvents list are prioritized from left to right, with the most preferred event listed to the left.

2) Persistent events that are not included in a RequestedEvents lists are less preferred than events (persistent or not) that are included in a RequestedEvents list. There is no well-defined preference order among persistent events that are not included in a RequestedEvents list.

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. 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.

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, and a new SignalRequests. However, the "Embedded NotificationRequest" cannot include another "Embedded NotificationRequest". When the "Embedded NotificationRequest" is activated, the list of observed events and the quarantine buffer will be unaffected (see 7.4.3.1).

The embedded NotificationRequest action allows the MGC 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 MGC 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 an MGC 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 wild-card "$" can be used to denote "the current connection"; however, this notation MUST NOT be used outside a connection handling command – the wild-card refers to the connection in question for the connection handling command.

The embedded ModifyConnection action allows the MGC 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. 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 attempted, and the previous successful connection mode changes in the list MUST remain effective.

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.

NOTE – Clause 7.4.3.1 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 is outside the scope of the core TGCP Recommendation. 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 Recommendation. 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 only a few exceptions to this rule, notably the fax and modem tones, which can be detected but cannot 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 MGC 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 7.4.3.1), i.e., events that have been detected by the gateway before the arrival of this NotificationRequest command, but have not yet been notified to the MGC. The parameter provides a set of handling options:

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

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

When the parameter is absent, the quarantined events MUST be processed. Support for the "lockstep" mode (via default) and "loop" mode is mandatory. An endpoint that receives a NotificationRequest with an unsupported QuarantineHandling parameter value SHOULD respond with error code 508 (unsupported QuarantineHandling).

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

DetectEvents is an optional parameter 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 7.4.3.1.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

7.3.2 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 7.1.1. 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 wild-card convention. A MGC that receives a Notification with wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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. Note that the MG MAY include only the domain name of its Notified Entity if only the domain name was received in the triggering NotificationRequest. The MGC SHOULD accept the value in this case. 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 (includes notification request embedded in connection handling primitives). 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", 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), and the final event that triggered the notification.

ReturnCode is a parameter returned by the MGC. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

7.3.3 CreateConnection

This command is used to create a connection.

```
ReturnCode
[, ConnectionId]
[, SpecificEndPointId]
[, LocalConnectionDescriptor]
       ← CreateConnection(CallId
       , EndpointId
              [, NotifiedEntity]
              [, LocalConnectionOptions]
              , Mode
              [, RemoteConnectionDescriptor]
              [, RequestedEvents]
              [, RequestIdentifier]
              [, 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. At a minimum, a Trunking Gateway MUST support one connection per endpoint. 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 MGCs 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-wild-carded value to the parameter EndpointId in the function call or it can be under-specified by using the "anyone" wild-card convention. If the endpoint is under-specified, the endpoint identifier will be assigned by the gateway and its complete value MUST be returned in the SpecificEndPointId parameter of the response only if the command is successful. In this case, the endpoint assigned MUST be in service and MUST NOT already have any connections on it. The "all" wild-card convention MUST NOT be used. A gateway that receives a CreateConnection with the "all" wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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 at least 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 8.4.1 for details. A list of permissible encoding methods is specified in ITU-T Rec. J.161. The literal names defined in 7.5 (Table 3)/J.161 MUST be used. Unknown compression algorithms SHOULD be ignored if they are received. See 7.7 for details on the codec selection process.

  NOTE – "encoding method" includes audio, image and video encodings.

- **Packetization Period**: A single packetization period in milliseconds MAY be specified with exactly one decimal value. If this specifier is used, then the same packetization period MUST be used for all encoding methods allowed by the LocalConnectionOptions. Note that if no encoding method field is specified in the LCO, the MG MUST NOT choose an encoding method with a packetization period that differs from that specified here. If different packetization periods for different encoding is desired, then this field MUST NOT be used. The value pertains to both media sent and received. Note that only the valid packetization period in conjunction with the associated encoding method are to be used by the MG. A list of permissible packetization periods is specified in the IPCablecom Audio/Video Codecs Specification (J.161). This specifier MUST NOT be supplied in the same LCO as the Multiple Packetization Period field. An MG MUST return an error (error code 524 – inconsistency in LocalConnectionOptions) when it receives an LCO with both the Packetization Period and Multiple Packetization Period fields.

- **Multiple Packetization Period**: A list of packetization periods in milliseconds MAY be specified if, and only if, the Encoding Method field is included. When specified, the multiple packetization period in milliseconds MUST contain exactly one decimal value or a hyphen for each entry in the encoding method field included in the LocalConnectionOptions. This applies even if several of the encoding methods have the same value. The first entry in the list MUST be a decimal number. When a hyphen is used, the codec in question MUST use the same packetization period as one of the other entries in the list that actually contains a decimal number, and furthermore the codec MUST NOT consume any more bandwidth than the other entry. This can for example be used for non-voice codecs (e.g., telephone-event or comfort noise) that use the same packetization period as the voice codec with which they are being used. Successive entries in the list of packetization periods MUST be ordered identically to the corresponding encoding methods. The values pertain to both media sent and received. Note that the MG MUST NOT choose a codec with a packetization period that differs from that specified here. Note that only the valid packetization period in conjunction with the associated encoding method are to be used by the MG. A list of permissible packetization periods is specified in the IPCablecom Audio/Video Codecs Specification (J.161). This specifier MUST NOT be supplied in the same LCO as the Packetization Period field. An MG MUST return an error (error code 524 – inconsistency in LocalConnectionOptions) under the following conditions:

  – When it receives an LCO with both the Packetization Period and Multiple Packetization Period fields.

– When it receives an LCO where the number of codecs specified in the Encoding Method field is different from the number of elements in the Multiple Packetization Period field.

- **Echo Cancellation**: Whether echo cancellation should be used initially on the line side or not[11]. 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 media gateway MUST apply echo cancellation initially. The media gateway SHOULD subsequently enable or disable echo cancellation in accordance with ITU-T Rec. V.8 when voiceband data is detected. For re-enabling echo cancellation, see ITU-T Rec. G.168. Following termination of voiceband data, the handling of echo cancellation MUST revert to the current value of the echo cancellation parameter. It is RECOMMENDED that echo cancellation handling is left to the gateway rather than having this parameter specified by the MGC.
- **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 0x00 (unless provisioned otherwise) MUST be used. When this parameter is present and valid, the endpoint MUST use the value provided to populate the differentiated services codepoint (DSCP) parameter in the IP header (see RFC 2474 for more information on the DSCP). The parameter value MUST be 0x00, or MUST be a multiple of four in the range of 0x01 to 0xFF (bits 6 and 7, the ECN bits, are reserved and hence must be set to "00"). An endpoint MUST return an error (error code 532 – Unsupported value(s) in LocalConnectionOptions) when it receives an invalid value. The left-most "bit" in the parameter corresponds to the most significant bit in the IP header.
- **Silence Suppression**: The telephony gateways may perform voice activity detection, and avoid sending packets during periods of silence. However, it is necessary for certain call types (e.g., modem calls) to disable silence suppression. 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 value is "off". If the value is "on", upon detecting voiceband data, the endpoint SHOULD disable silence suppression. Following termination of voiceband data, the handling of silence suppression MUST revert to the current value of the silence suppression parameter.

Additionally, the following LocalConnectionOptions fields are used to support the IPCablecom security services: (Either or both ciphersuites MAY be present):

- **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 according to the rules described in the IPCablecom Security Specification (J.170). The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives (see 8.4.1 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 is specified in ITU-T Rec. J.170. The RTP ciphersuite parameter applies to RTP media streams only. If the MGC includes LocalConnectionOptions that require use of non-RTP media only (as for example T.38 fax relay using UDPTL), the RTP ciphersuite parameter MUST NOT be included. If an LCO allows both RTP and non-RTP media and an RTP ciphersuite parameter is included, it applies to the RTP media only. In all cases, if the resulting media stream on the connection

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

is not an RTP media stream (as for example T.38 fax relay using UDPTL), the RTP ciphersuite parameter MUST be ignored, i.e., RTP security is not being used, and RTP security parameters are not included in LocalConnectionDescriptor.

- **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 according to the rules described in the IPCablecom Security Specification (J.170). The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives. See 8.4.1 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 is specified in ITU-T Rec. J.170. The RTCP ciphersuite parameter applies to RTCP for RTP media streams only. If the MGC includes LocalConnectionOptions that require use of non-RTP media only (as for example T.38 fax relay using UDPTL), the RTCP ciphersuite parameter MUST NOT be included. If an LCO allows both RTP and non-RTP media and an RTCP ciphersuite parameter is included, it applies to RTCP for the RTP media only. In all cases, if the resulting media stream on the connection is not an RTP media stream (as for example T.38 fax relay using UDPTL), the RTCP ciphersuite parameter MUST be ignored, i.e., RTCP security is not being used, and RTCP security parameters are not included in LocalConnectionDescriptor.

The trunking gateway 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.

Furthermore, TGCP supports IPCablecom Electronic Surveillance. When a connection is subject to electronic surveillance, all valid media packets received on the connection and all media packets sent on the connection will be replicated and forwarded to an Electronic Surveillance Delivery Function[12] after inclusion of a Call Content Connection Identifier. The replication will follow the connection mode for the connection, except for media generated by signals applied to the connection, which will be replicated regardless of the connection mode. For example, a connection in "inactive" mode will not generate any intercepted media[13], whereas a connection in "sendonly" mode will only generate intercepted media in the send direction. Replicated packets will not be included in statistics for the connection. The following LocalConnectionOptions fields are used to support IPCablecom Electronic Surveillance:

- **Call Content Connection Identifier**: The Call Content Connection (CCC) Identifier is a 32-bit value that specifies the Call Content Connection Identifier to be used for connections that are subject to electronic surveillance. It will be added to the header of intercepted voice packets.
- **Call Content Destination**: The Call Content Destination specifies an IPv4 address followed by a colon and a UDP port number. The Call Content Destination specifies the destination IP address and port for the call content intercepted.

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 8.4 details the supported use of SDP in the TGCP 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

---

[12] Note that the replication occurs at the network level – see PKT-SP-ESP-I01-991229 for details.

[13] Assuming no media generating signal was applied to the connection.

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.

When codecs are changed during a communication, small periods of time may exist where the endpoints use different codes. As stated above, trunking gateways 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", "inactive", "loopback", "continuity test", "network loopback" or "network continuity test". The handling of these modes is specified in the beginning of 7.3. Note that signals applied to a connection do not follow the connection mode. Some endpoints may not be capable of supporting all modes – see V.1. 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", "network loopback", "network continuity test", or if a signal (as opposed to detecting an event) is to be applied to the connection (error code 527 – missing RemoteConnectionDescriptor is RECOMMENDED).

ConnectionId is a parameter returned by the gateway that uniquely identifies the connection within the context of the endpoint in question. The parameter MUST be included with any provisional or successful response to a CreateConnection command. The parameter MUST NOT be included when any error response is returned and the connection was not created.

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 8.4 details the supported use of SDP in the TGCP profile. The LocalConnectionDescriptor parameter MUST be included with any provisional or successful response to a CreateConnection command. The LocalConnectionDescriptor MUST NOT be included when any error response is returned and the connection was not created.

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 MGC. 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", "loopback", or "continuity test", 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", or "send/receive".

The **RequestedEvents**, **RequestIdentifier**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are all optional. They can be used by the MGC to effectively include an embedded 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. Note that absence of the RequestedEvents and SignalRequests parameters is interpreted as an empty list only if a RequestIdentifier parameter is included.

As an example of use, consider an MGC that wants to place a call to an operator services system through an MF trunking gateway. The MGC could:

- ask the trunking gateway to create a connection, in order to be sure that the media gateway has resources for the call;
- ask the trunking gateway to seize an MF operator services trunk and initiate the call;
- ask the trunking gateway to notify the MGC when the call has been placed.

All of the above can be accomplished in a single CreateConnection command by including a notification request with the RequestedEvents parameters for the answer event and the SignalRequests parameter for the set-up 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 call initiation notification request must be refused in the glare condition if the circuit is already seized. In this example, the call must not be placed if the connection cannot be established, and the connection must not be established if the circuit is already seized. An error would be returned instead (error code 401 – circuit already seized), which informs the MGC 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 7.5) optionally followed by commentary.

7.3.4 ModifyConnection

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

```
ReturnCode
[, LocalConnectionDescriptor]
        ← ModifyConnection(CallId
                , EndpointId
                , ConnectionId
                [, NotifiedEntity]
                [, LocalConnectionOptions]
                [, Mode]
                [, RemoteConnectionDescriptor]
                [, RequestedEvents]
                [, RequestIdentifier]
                [, 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 wild-card convention. A gateway that receives a ModifyConnection with a wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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

- Provide information on the other end of the connection through the **RemoteConnectionDescriptor**.
- Activate or deactivate the connection by changing the **mode** parameter's value. This can occur at any time during the connection, with arbitrary parameter values. An activation can, for example, be set to the "receive only" mode.
- Change the parameters of the connection through the **LocalConnectionOptions**, for example, by switching to a different coding scheme, changing the packetization period, or modifying the handling of echo cancellation.

The command will only return a **LocalConnectionDescriptor** if the local connection parameters, such as, 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. The **LocalConnectionDescriptor** parameter MUST NOT be included when any error response is returned and the connection was not modified. 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[14].

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 MGC[15]. When RTP address information is given to a trunking gateway for a connection, the trunking gateway 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. ITU-T Rec. J.170 should be consulted for additional security requirements.

The **RequestedEvents**, **RequestIdentifier**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are optional. The parameters can be used by the MGC to include an embedded 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. Note that absence of the RequestedEvents and SignalRequests parameters is interpreted as an empty list only if a RequestIdentifier parameter is included.

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.

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

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

---

[14] 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.

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

7.3.5 DeleteConnection (from the media gateway controller)

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]
               [, SignalRequests]
               [, QuarantineHandling]
               [, DetectEvents])
```

The endpoint identifier, in this form of the DeleteConnection command, MUST be fully qualified. Wild-card conventions MUST NOT be used. A gateway that receives this form of the DeleteConnection with a wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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

In response to the DeleteConnection command, the gateway returns a list of parameters that describe the status of the connection[16]. The connection parameters MUST only be returned if the command is successful and the connection is deleted. 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 MUST be based on the same information provided via the RTCP mechanism.
- **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 MUST be based on the same information provided via the RTCP mechanism.
- **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. All received packets MUST be counted independent of the connection mode or any type of processing error, e.g., authentication failure.
- **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. All received packets MUST be counted independent of the connection mode or any type of processing error, e.g., authentication failure.

---

16 The values calculated will not include packets that resulted from Electronic Surveillance.

- **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, for example, no packets were received on the connection.
- **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 IETF RFC 1889. The count includes packets received from different SSRC if the sender used several values. The value is zero if, for example, no packets were received on the connection.
- **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. Trunking gateway 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 IETF RFC 1889.

The **RequestedEvents**, **RequestIdentifier**, **SignalRequests**, **QuarantineHandling**, and **DetectEvents** parameters are optional. They can be used by the MGC to transmit an embedded 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 circuit is disconnected, the gateway might be instructed to delete the connection and to start looking for a seizure event. This can be accomplished in a single DeleteConnection command also by transmitting the RequestedEvents parameter for the seizure event and an empty SignalRequests parameter. Note that absence of the RequestedEvents and SignalRequests parameters is interpreted as an empty list only if a RequestIdentifier parameter is included.

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.

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

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

7.3.6 DeleteConnection (From the trunking gateway)

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. Wild-card conventions MUST NOT be used. A MGC that receives a DeleteConnection with a wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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 7.6.

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

ReturnCode is a parameter returned by the MGC. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

7.3.7 DeleteConnection (Multiple Connections From the Media Gateway Controller)

A variation of the DeleteConnection function can be used by the MGC 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" wild-card. All connections for the endpoint(s) with the CallId specified will be deleted. The command does not return any individual statistics or call parameters. A gateway that receives a DeleteConnection (Multiple Connections From the Media Gateway Controller) with the "any of" wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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

```
ReturnCode
     ← DeleteConnection(EndpointId)
```

In this form of the DeleteConnection command, MGCs 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" wild-carding convention, as specified in 7.1.1. The "any of" wild-carding 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. Also after the connection has been deleted, any loopback that has been requested for the connection MUST be cancelled (unless the endpoint has another connection requesting loopback).

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

7.3.8 Auditing

The MGCP is based upon a centralized call control architecture where a MGC acts as the remote controller of client devices that provide voice communications 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 trunking gateways and the MGCs in an IPCablecom system is provided to allow the MGC to audit endpoint and connection state and to retrieve protocol-specific capabilities of an endpoint.

Two commands for auditing are defined for the trunking gateways:

AuditEndPoint: Used by the MGC to determine the status of an endpoint.

AuditConnection: Used by the MGC 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 trunking gateway 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 trunking gateway, both of which are outside the scope of this Recommendation.

7.3.8.1 AuditEndPoint

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

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

The **EndpointId** identifies the endpoint that is being audited. The "any of" wild-card convention MUST NOT be used. A gateway that receives an AuditEndPoint with the "any of" wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

The "all of" wild-card 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 wild-card 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 wild-card 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, SignalRequests, RequestIdentifier, NotifiedEntity, ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, VersionSupported, MaxMGCPDatagram, and Capabilities.

If an endpoint is queried about a parameter it does not support, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response.

If an endpoint is queried about a parameter it does support, but has no value for, the endpoint MUST NOT generate an error; instead the parameter MUST be included in the response with an empty parameter value.

Only when successful, the AuditEndPoint response MUST, in turn, include information about each of the items for which auditing information was requested. Note that parameters that are explicitly marked "optional" below and not supported by the endpoint are omitted in the response:

- **RequestedEvents**: The current value of RequestedEvents the endpoint is using including the action associated with each event. Persistent events are included in the list.
- **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[17]. 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. Note that the MG MAY include only the domain name of its Notified Entity if only the domain name was provided to it via the Notified Entity parameter of a TGCP message or acknowledgement. The MGC SHOULD accept the value in this case.
- **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., seizure if the MF trunk for the endpoint is currently seized. The definition of the individual events will state if the event in question has an auditable state associated with it.

[17] Currently, there should be no pending Brief signals.

- **VersionSupported**: A list of protocol versions supported by the endpoint.
- **MaxMGCPDatagram**: The maximum size of an MGCP datagram in bytes supported by the endpoint (see 8.5.3). The value excludes any lower layer overhead. Support for this parameter is optional. The default maximum MGCP datagram size is assumed if a value is not returned.
- **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, e.g., silence suppression, are only compatible with some codecs, then the gateway will return several capability sets.
  - **Compression Algorithm** — A list of supported codecs. The literal names defined in clause 7.5 (Table 3) of the IPCablecom Audio/Video Codecs Specification (J.161) MUST be used. Unknown compression algorithms SHOULD be ignored if they are received. 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[18].
  - **Silence Suppression** — Whether silence suppression is supported or not.
  - **Type of Service** — Whether type of service is supported or not.
  - **Event Packages** — A list of event packages supported. The first event package in the list will be the default package.
  - **Modes** — A list of supported connection modes.
  - **RTP Ciphersuites** — A list of authentication and encryption algorithms supported for RTP.
  - **RTCP Ciphersuites** — A list of authentication and encryption algorithms supported for RTCP.
  - **Electronic Surveillance** — Whether IPCablecom Electronic Surveillance is supported or not.

The MGC 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 7.5) 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 seizure state may change before the MGC receives the above information.

[18] Currently, all TGCP endpoints must support echo cancellation.

7.3.8.2 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-wild-cards MUST NOT be used. A gateway that receives an AuditConnection with a wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response. 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.

If an endpoint is queried about a connection parameter it does not support, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response.

If an endpoint is queried about a connection parameter it does support, but has no value for, the endpoint MUST NOT generate an error; instead the parameter MUST be included in the response with an empty parameter value.

Only when successful, the AuditConnection response MUST, in turn, include information about each of the items for which auditing information was requested. Note that parameters that are explicitly marked "optional" below and not supported by the endpoint are omitted in the response:

- **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 most recent RemoteConnectionDescriptor that was supplied in a previous CreateConnection or ModifyConnection command to the gateway for this connection.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see 7.5) 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).

7.3.9 RestartInProgress

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" wild-card 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" wild-card convention MUST NOT be used. A MGC that receives a Restart in Progress with the "any of" wildcard convention MUST return an error (error returned SHOULD be error code 500 – the transaction could not be executed because the endpoint is unknown) in response.

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 MGC should refrain from establishing new connections, and should try to gracefully tear down any existing connections. At the expiry of the restart delay, the MG should send a new RSIP message with a restart method of "forced". This will explicitly indicate to the MGC that the endpoints are now out of service.
- A "cancel-graceful" restart method indicates that a gateway is cancelling a previously issued "graceful" restart method for the same endpoints. The endpoints remain in service. When this command is sent, the gateway will immediately begin to allow the establishment of new connections on these endpoints.
- 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 endpoint will never go out of service as a result of this operation, and that the MGC should simply wait for the natural termination of the existing connections, without establishing new connections. The restart delay is always considered null in the case of the "forced" and "cancel-graceful" methods and hence the "restart delay" parameter MUST NOT be used with these restart methods by the gateway. A restart delay of null for the "restart" method indicates that service has already been restored. This typically will occur after gateway startup/reboot. To mitigate the effects of a gateway IP address change, the MGC MAY wish to resolve the gateway's domain name by querying the DNS regardless of the TTL of a current resource record for the restarted gateway.

Trunking gateways SHOULD send a "graceful" or "forced" RestartInProgress message as a courtesy to the MGC when they are taken out of service, e.g., by being shut down. The Call Agent cannot rely on receiving such messages. However, a trunking gateway MUST send a "forced" or "graceful" RestartInProgress message when an endpoint is removed from service through the

provisioning process or upon detection of a failure of the endpoint transport facility (e.g., loss of signal, receipt of yellow alarm, etc.). Trunking gateways MUST send a "restart" RestartInProgress message with a null delay to their MGC when they are back in service according to the restart procedure specified in 7.4.3.5 – MGCs can rely on receiving this message. Also, trunking gateways MUST send a "disconnected" RestartInProgress message to their current "notified entity" according to the "disconnected" procedure specified in 7.4.3.6. The "restart delay" parameter MUST NOT be used with the "forced" and "cancel graceful" restart methods.

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

ReturnCode is a parameter returned by the MGC. It indicates the outcome of the command and consists of an integer number (see 7.5) optionally followed by commentary.

A **NotifiedEntity** MAY additionally be returned with the response to the RestartInProgress from the Media Gateway Controller – this should normally only be done in response to "restart" or "disconnected" (see also 7.4.3.5 and 7.4.3.6). If a NotifiedEntity parameter was included in the response returned, it specifies a new "notified entity" for the endpoint(s) – this operation SHOULD only be done with the response error code 521 (endpoint redirected). Note that the above behaviour for returning a NotifiedEntity in the response is only defined for RestartInProgress responses and SHOULD NOT be done for responses to other commands. Any other behaviour is undefined:

- If the response indicated success (return code 200 – transaction executed), the restart in question completed successfully, and the NotifiedEntity returned is the new "notified entity" for the endpoint(s).
- If the response from the MGC indicated an error code, the restart in question is not yet complete. If the response was 521 (endpoint redirected), then the response MUST include a NotifiedEntity parameter which specifies the new "notified entity" for the endpoint(s), and MUST be used when retrying the restart in question (as a new transaction).

In the case of "restart" and "disconnected", the restart in question MUST be retried whenever the Media Gateway Controller returns a transient error code (4xx), whereas it SHOULD be retried for any other restartMethod. It is RECOMMENDED that any type of restart is terminated if a permanent error code (5xx) is returned, except for error code 521, as specified above.

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

7.4 States, failover and race conditions

In order to implement proper call signalling, the MGC must keep track of the state of the endpoint, and the gateway must make sure that events are properly notified to the MGC. Special conditions may exist when the gateway or the MGC are restarted: the gateway may need to be redirected to a new MGC during "failover" procedures; similarly, the MGC may need to take special action when the gateway is taken offline, or restarted.

7.4.1 Recaps and highlights

As mentioned in 7.1.4, MGCs 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 failover in MGCP:

- An MGC 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, MGC associated with it at any given point in time. The MGC associated with an endpoint is the current value of the "notified entity".
- The "notified entity" is initially set to a provisioned value. When commands with a NotifiedEntity parameter is received for the endpoint, including wild-carded endpoint-names, the "notified entity" is set to the value specified. If the "notified entity" for an endpoint is empty or has not been set explicitly[19], 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 MGC will thus be identified by its network address, which SHOULD only be done on exceptional basis.
- Responses to commands are always sent to the source address of the command, regardless of the current "notified entity". When a Notify message needs to be 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. An MGC can however instruct them to switch by providing them with a new "notified entity".
- If an MGC becomes unavailable, the endpoints managed by that MGC will eventually become "disconnected". The only way for these endpoints to become connected again is either for the failed MGC to become available again, or for another (back-up) MGC to contact the affected endpoints with a new "notified entity".
- When another (back-up) MGC has taken over control of a group of endpoints, it is assumed that the failed MGC will communicate and synchronize with the back-up MGC in order to transfer control of the affected endpoints back to the original MGC, if so desired. Alternatively, the failed MGC could simply become the back-up MGC now.

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

7.4.2 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 8.5), commands are repeated. Gateways MUST keep in memory a list of the responses that they sent to recent transactions, (a list of all the responses they sent over the last Tt_{hist} seconds) and a list of the transactions that have not yet finished executing. 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; subsequent handling depends on the command in question. If it is a CreateConnection or ModifyConnection command, the gateway MUST send a provisional response. If it is any other command, it is simply ignored. In either case, a final response will be provided when the execution of the command is complete.

---

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

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 an MGC becomes unavailable;
- MGC failure, when, e.g., all interfaces for a MGC becomes unavailable;
- failover, when a new MGC 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 an MGC 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 an MGC 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. It should be noted that the "suspicion threshold", which is identified as "Max1", is normally lower than the "disconnection threshold", which is identified as "Max2", and MUST be set to a larger value than Max 1.

The MGCP retransmission algorithm is illustrated in Figure 2 and explained further in the following:

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after retransmitting the packet an excessive number of times (typically between 7 and 11 times). In order to account for the possibility of an undetected or in-progress "failover", we modify the classic algorithm as follows:

- The gateway MUST always check for the presence of a new MGC. It can be noticed by:
  - receiving a command where the NotifiedEntity points to a new MGC; or
  - receiving a redirection response pointing to a new MGC.
- If a new MGC is detected, the gateway MUST direct retransmissions of any outstanding commands for the endpoint(s) redirected to that new MGC. 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, then retransmissions MUST cease. If more than $2*Tt_{hist}$ has elapsed, then the endpoint becomes disconnected.
- If the number of retransmissions to this MGC equals "Max1", the gateway MAY actively query the name server in order to detect the possible change of MGC interfaces, regardless of the Time To Live (TTL) associated with the DNS record.
- The gateway may have learned several IP addresses for the MGC. If the number of retransmissions for this IP address is greater than or equal to "Max1" and less 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. Also, receipt of explicit network notifications such as, e.g., ICMP network, host, protocol, or port unreachable SHOULD lead the gateway to try alternate addresses (with due consideration to possible security issues).
- If there are no more interfaces to try, and the number of repetitions for this address is "Max2", then the gateway SHOULD contact the DNS one more time to see if any other interfaces have become available. If, there still are no more interfaces to try, then

retransmissions MUST cease. If more than 2*Tt_{hist} has elapsed, then the endpoint becomes disconnected.

- Once an endpoint becomes disconnected, subsequent processing depends on whether the loss of association was detected by the gateway or Media Gateway Controller, as follows:
  - The gateway MUST initiate the "disconnected" procedure as specified in 7.4.3.6.
  - The Media Gateway Controller MUST NOT attempt to use the endpoint for any new calls until connectivity has been restored. Furthermore, the Media Gateway Controller MUST implement an algorithm to detect when connectivity with the endpoint is subsequently restored (e.g., on receiving a response to a periodic Audit Endpoint command). When connectivity to the endpoint is restored, and if no other condition exists which prevents the endpoint from supporting calls, the MGC MUST ensure that the endpoint can be used for new calls without requiring any manual intervention.

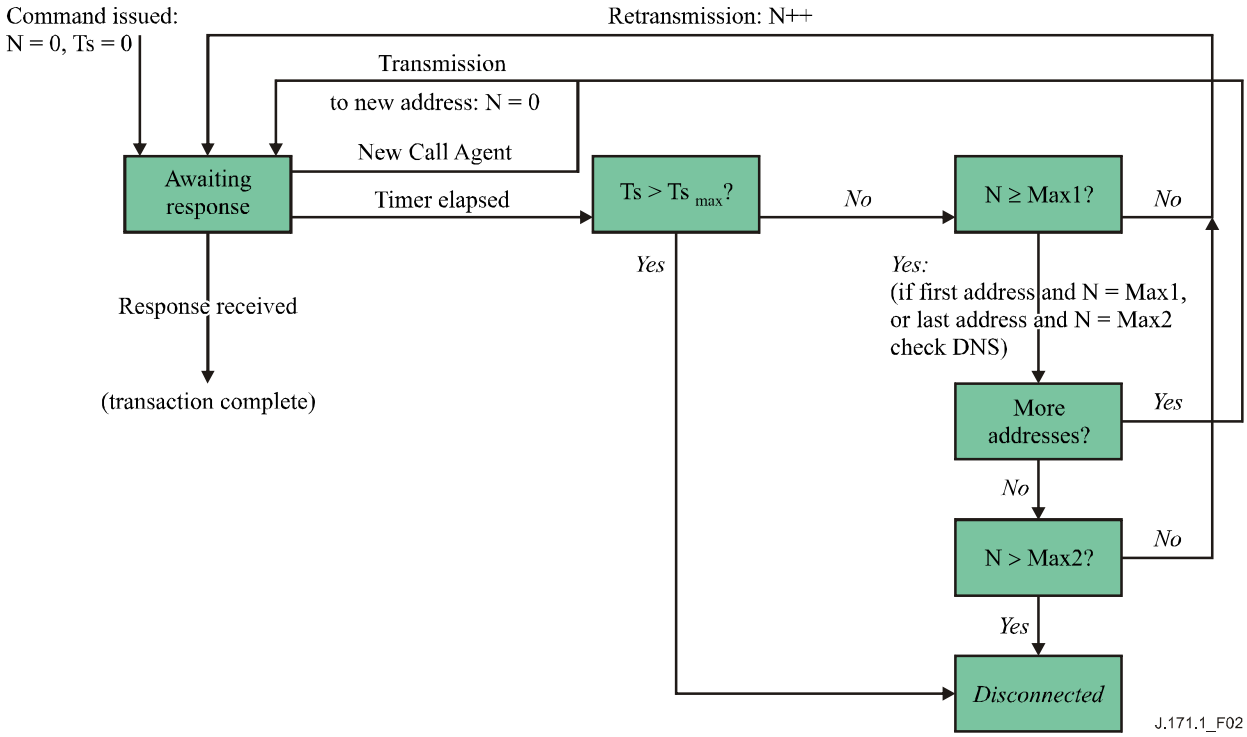


Figure 2/J.171.1 – Retransmission algorithm

In order to adapt to network load automatically, MGCP specifies exponentially increasing timers (see 8.5.2). 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 is however important that the maximum delay of retransmissions be bounded. Prior to any retransmission, it is checked that the time (Ts) elapsed since the sending of the initial datagram is no greater than Ts_{max}. If more than Ts_{max} time has elapsed, retransmissions MUST cease. When Ts_{max} has expired, or all of the retransmissions to all known IP addresses have been sent, there is a pause before declaring the endpoint disconnected. This pause represents a period of time where the only action is to wait for a response from any of the recent retransmissions. The quiescent period lasts for what remains of twice the life expectancy of the original transaction (2*Tt_{hist}). This settling time allows all active transactions to complete or time out before the endpoint is declared disconnected.

This helps ensure that each restart of the endpoint is from a clean and initial state. If more than 2 * Tt_{hist} time has elapsed, the endpoint becomes disconnected. The value Ts_{max} is related to the Tt_{hist} value: the Tt_{hist} value MUST be greater than or equal to Ts_{max} plus the maximum propagation delay in the network, Tp_{max}.

In other words, 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.

7.4.3 Race conditions

This clause describes 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 seizure-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 MGC 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.

7.4.3.1 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, and the "Detect Events" list.

When the endpoint is initialized, the requested events list only consists of persistent events for the endpoint. 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. The events that are defined as "accumulate" are accumulated in a list of observed events. 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[20]. The Notify command may be retransmitted in the "notification state", as specified in 7.4.2.

If the endpoint is or has become disconnected (see 7.4.2) in the meantime, a response to the Notify command will never be received. The Notify command is then lost and hence no longer considered pending, yet the endpoint is still in the "notification state". Should this occur, completion of the disconnected procedure specified in 7.4.3.6 shall then lead the endpoint to exit the "notification state".

When the endpoint exits the "notification state", it resets the list of observed events of the endpoint to a null value. Following that point, the behaviour of the gateway depends on the value of the QuarantineHandling parameter in the triggering NotificationRequest command:

If a QuarantineHandling parameter of "step" was specified by the Call Agent/Media Gateway Controller or if no value was specified, then the gateway will use "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".

If a QuarantineHandling parameter of "loop" was specified by the Call Agent/Media Gateway Controller (i.e., "loop" mode), it will proceed as follows. When the gateway exits the "notification state", it resets the list of observed events of the endpoint to a null value and starts processing the list of quarantined events, using the already received list of requested events. When processing these events, the gateway may encounter an event, which triggers a Notify command to be sent. If that is the case, the gateway can adopt one of the two following behaviours:

– it can immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included, leaving the unprocessed events in the quarantine buffer;
– it can attempt to empty the quarantine buffer and transmit a single Notify command reporting several sets of events. The events that follow the last triggering event MUST be left in the quarantine buffer.

If the gateway transmits a Notify command, the endpoint will re-enter and remain in the "notification state" until the acknowledgement is received (as described above). If the gateway does not find a quarantined event that triggers a Notify command, it places the endpoint in a normal state. Events are then processed as they come, in exactly the same way as if a Notification Request command had just been received.

A gateway can receive at any time a new NotificationRequest command for the endpoint, including the case where the endpoint is disconnected, which will also have the effect of taking the endpoint out of the "notification state" assuming the NotificationRequest executes successfully. Activating an embedded NotificationRequest is here viewed as receiving a new NotificationRequest as well, except that the current list of ObservedEvents remains unmodified rather than being processed again.

When a new NotificationRequest is received in the "notification state", the gateway SHOULD attempt to deliver the pending Notify (note that a Notify that was lost due to being disconnected is no longer considered pending) 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

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

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/MGC 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;
   - Notify's for a given endpoint MUST be delivered in-order. 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".

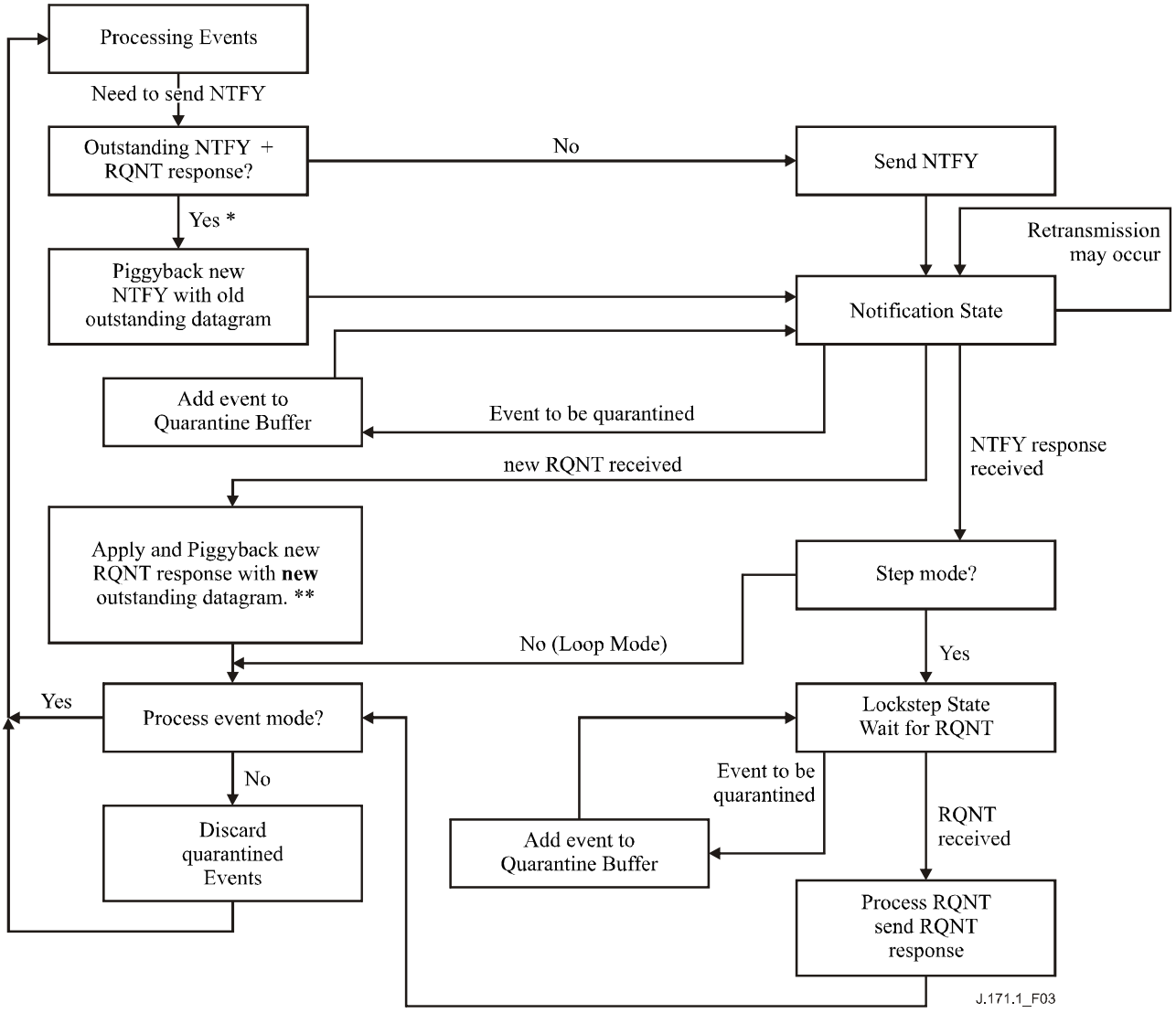
After receiving a NotificationRequest command, the "requested events" list is replaced by the newly received parameters. Furthermore, when the NotificationRequest was received in the "notification state", the list of "observed events" is reset to a null value. The subsequent behaviour is then conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events and observed events (which in this case is an empty list), are to be discarded, in which case all quarantined and observed events are discarded. If the parameter specifies that the quarantined and observed events should be processed, the gateway will start processing the list of quarantined and observed events, using the newly received list of "requested events". 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.

A new notification request may be received while the gateway has accumulated events according to the previous notification requests, but has not yet detected any notification-triggering events. The handling of not-yet-notified events is determined, as with the quarantined events, by the quarantine handling parameters:

- If the quarantine-handling parameter specifies that quarantined events shall be ignored, the observed events list is simply reset.
- If the quarantine-handling parameter specifies that quarantined events shall be processed, the observed event list is transferred to the quarantined event list. The observed event list is then reset, and the quarantined event list is processed. The only exception is the activation of an embedded Notification Request. In this case, the observed event list remains unmodified rather than being processed 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.

Figure 3 illustrates the procedure specified above assuming all transactions execute successfully:



* This decision leg is taken if the gateway needs to send a new Notify while it is waiting for a response to a previous Notify on the same endpoint. This could occur as a result of receiving a new RQNT in the "Notification State", as described in the text accompanying the diagram.

** The "new outstanding datagram" refers to the datagram containing the pending Notify, possibly piggy-backed with additional Notify(s) and RQNT response(s), that was being retransmitted in the "notification state" when the new RQNT was received. The in-order delivery of the RQNT response with pending Notify(s) is optional; the gateway may choose instead to send the RQNT response in a separate datagram. The requirement to ensure in-order delivery of Notify's, on the other hand, is mandatory.

Figure 3/J.171.1 – Quarantine algorithm

MGCs SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggybacking mechanism[21].

7.4.3.2 Explicit detection

A key element of the state of several endpoints is the seizure state of a circuit. Although seizure-state changing events are persistent in TGCP, race conditions and state mismatch may still occur, for example when a circuit is seized while the MGC is in the process of requesting the gateway to look for seizure (the "glare" condition well known in telephony – this is however primarily an issue for two-way CAS trunks, which are not supported in this version of this Recommendation).

---

[21] Vendors that choose not to follow this Recommendation should examine Media Gateway Controller failure scenarios carefully.

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 a "seizure"[22] transition while the circuit is already seized (error code 401 – circuit seized);

2) if the gateway is requested to notify an "unseize"[23] condition while the circuit is not seized (error code 402 – circuit not seized).

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

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

The other state variables of the gateway, such as the list of requested events or list of requested signals, are entirely replaced after each successful NotificationRequest, which prevents any long-term discrepancy between the MGC 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.

When the Media Gateway Controller receives an error response indicating that a NotificationRequest was unsuccessful, it MUST take action to ensure that any events quarantined by the endpoint are processed or discarded, and that the endpoint is returned to a normal operating mode where newly requested events are reported as they occur. For example, say the Media Gateway Controller receives an error code "401 – endpoint already seized" response to a NotificationRequest requesting detection of the "MT/sup" signal. At this point the Media Gateway Controller should assume that the NotificationRequest had no effect on the endpoint, and that the endpoint is in the same state that it was prior receiving the command. If the endpoint was quarantining events in the "lockstep" state prior to receiving the NotificationRequest, then it will still be quarantining events in the "lockstep" state after sending the "401" error response. To ensure that the endpoint is not left in a state where it is permanently quarantining events, the Media Gateway Controller should send a new NotificationRequest with a different (possibly empty) set of requested events to transition the endpoint out of the "lockstep" state and into the normal mode where it can report new events.

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 (includes notification request embedded in connection handling primitives) commands thereby enabling the MGC to determine if the Notify command was generated before or after the gateway received the new NotificationRequest.

7.4.3.3 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 seizure-state, must be defined carefully.

---

[22] For instance by requesting the "sup" event on an MF Terminating BLV/OI trunk with a call already in progress.

[23] For instance by requesting the "rel" event on an MF Operator Services trunk without any call in progress.

An important point to consider is that the seizure-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 preconditions 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 preconditions 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 "seizure" event. When the transaction is initiated the circuit is "not seized" and this precondition is therefore met. If the seizure-state changes to "seized" before the transaction completes, the precondition is no longer met, and the transaction therefore immediately fails. The "seizure" event will now be stored in the "quarantine" buffer which then gets processed.

7.4.3.4 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 MGC 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.

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

1) When a gateway handles several endpoints, commands pertaining to the different endpoints can be sent in parallel, for example following a model where each endpoint is controlled by its own process or its own thread.
2) When several connections are created on the same endpoint, commands pertaining to different connections can be sent in parallel.
3) On a given connection, there should normally be only one outstanding command (create or modify). However, a DeleteConnection command can be issued at any time. In consequence, a gateway may sometimes receive a ModifyConnection command that applies to a previously deleted connection. Such commands MUST be ignored, and an error returned (error code 515 – incorrect connection-id).
4) On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter is used to correlate Notify commands with the triggering NotificationRequest.
5) In some cases, an implicitly or explicitly wild-carded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The MGC should individually delete all connections whose completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wild-carding should not be sent until a response to the wild-carded DeleteConnection command is received.
6) When commands are embedded within each other, sequencing requirements for all commands MUST be adhered to. For example, a CreateConnection command with a notification request in it must adhere to the sequencing requirements for CreateConnection and NotificationRequest at the same time.

7) AuditEndpoint and AuditConnection is not subject to any sequencing.

8) RestartInProgress must always be the first command sent by an endpoint as defined by the restart procedure (see 7.4.3.5). 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 trunking gateways; however, the trunking gateway MUST NOT make any assumptions as to whether MGCs follow the rules or not. Consequently gateways MUST always respond to commands, regardless of whether they adhere to the above rules or not.

To ensure consistent operation, trunking gateways SHOULD behave as specified below when one or more of the above rules are not followed:

- Where a single outstanding command is expected (ModifyConnection, NotificationRequest), but the same command is received in a new transaction before the old finishes executing, the gateway SHOULD fail the previous command. This includes the case where one or more of the commands were encapsulated. The use of error code 407 (transaction aborted) is RECOMMENDED.
- If a ModifyConnection command is received for a pending CreateConnection command, the ModifyConnection command SHOULD simply be rejected. The use of error code 400 (transient error) is RECOMMENDED. Note that this situation constitutes a Media Gateway Controller programming error.

Note, that where reception of a new command leads to aborting an old command, the old command SHOULD be aborted regardless of whether the new command succeeds or not. For example, if a ModifyConnection command is aborted by a DeleteConnection command which itself fails due to an encapsulated NotificationRequest, the ModifyConnection command is still aborted.

7.4.3.5 Fighting the restart avalanche

Suppose that a large number of gateways are powered on simultaneously. If they were to all initiate a RestartInProgress transaction, the MGC 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, or when all or a subset of the gateway's endpoints are returned to service, the gateway 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 MGC, or the detection of a local circuit activity, such as, for example, seizure transition on a trunking gateway. A pre-existing seizure condition results in the generation of a seizure event.

3) When the restart timer elapses, when a command is received, or when an activity or pre-existing seizure condition is detected, the gateway initiates the restart procedure.

The restart procedure simply states that the endpoint MUST send a RestartInProgress command to the MGC informing it about the restart and furthermore guarantee that the first message (command or response) that the MGC sees from this endpoint MUST be this RestartInProgress command. During each initiation of "disconnected" procedure, the command MUST observe the normal retransmission and transaction identifiers requirements (see 7.4.2).

The endpoint MUST take full advantage of piggybacking in achieving this. For example, if a circuit seizure activity occurs prior to the restart timer expiring, a packet containing the RestartInProgress command, and with a piggybacked Notify command for the seizure event will be generated. In the case where the restart timer expires without any other activity, the gateway simply sends a RestartInProgress message.

The restart procedure is complete once a success response has been received. If an error response is received, the subsequent behaviour depends on the error code in question:

- If the error code indicates a transient error (4xx), then the restart procedure MUST be initiated again (as a new transaction).
- If the error code is 521, then the endpoint is redirected, and the restart procedure MUST be initiated again (as a new transaction). The 521 response should have included a NotifiedEntity which then is the "notified entity" towards which the restart is initiated.
- If the error is any other permanent error (5xx), then it is RECOMMENDED that the endpoint no longer initiates the restart procedure on its own (until rebooted) unless otherwise specified. If a command is received, the endpoint MUST initiate the restart procedure again.

Note that if the RestartInProgress is piggy-backed with the response (R) to a command received while restarting, then retransmission of the RestartInProgress does not require piggy-backing of the response R. However, while the endpoint is restarting, a resend of the response R does require the RestartInProgress to be piggy-backed to ensure in-order delivery of the two.

Should the endpoint enter the "disconnected" state while carrying out the restart procedure, the disconnected procedure specified in 7.4.3.6 MUST be carried out, and a "disconnected" message is sent during the procedure.

It is expected that each endpoint in a gateway will have a provisionable MGC, 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 MGC, the above procedure must be performed for each collection of endpoints managed by a given MGC. The gateway MUST take full advantage of wild-carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same MGC.

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 a gateway.

MGCs are typically dimensioned to handle the peak-hour traffic load, during which, on average, 60% of the trunks will be busy serving calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 transactions between each endpoint and the MGC. This simple calculation shows that the MGC is expected to handle 5 to 6 transactions for each endpoint, every 5 minutes on average, or, to put it otherwise, about one transaction per endpoint per minute. This suggests that a reasonable value of MWD would be 2 minutes per endpoint. When the value of MWD is set for the gateway, the value should be inversely proportional to the number of endpoints that are being restarted. For example MWD should be set to 5 seconds for a gateway that handles a T1 line, or to 180 milliseconds for a gateway that handles a T3 line.

7.4.3.6 Disconnected endpoints

In addition to the restart procedure, trunking gateways also have a "disconnected" procedure, which is initiated when an endpoint becomes "disconnected" as described in 7.4.2. It should here be noted that endpoints can only become disconnected when they attempt to communicate with the MGC. 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 MGC, or the detection of a local circuit activity for the endpoint, such as for example a seizure transition.

3) When the "disconnected" timer elapses, when a command is received, or when a local circuit activity is detected, the gateway MUST initiate the "disconnected" procedure with a new transaction ID for the endpoint. In the case of local circuit activity, a provisionable "disconnected" minimum waiting delay (Td_{min}) must furthermore have elapsed since the gateway became disconnected or the last time it ended 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 MGC informing it that the endpoint was disconnected and furthermore guarantee that the first message (command or response) that the MGC now sees from this endpoint MUST be this RestartInProgress command. During each initiation of "disconnected" procedure, the command MUST observe the normal retransmission and transaction identifiers requirements (see 7.4.2). The endpoint MUST take full advantage of piggybacking in achieving this.

On receiving a RestartInProgress message containing a restart method of "disconnected", the Media Gateway Controller MUST take action to ensure that any events quarantined by the endpoint are processed or discarded, and that the endpoint is returned to a normal operating mode where newly requested events are reported as they occur. The Media Gateway Controller SHOULD send a NotificationRequest containing a QuarantineHandling parameter set to "discard" in this case. The Media Gateway Controller may also decide to perform one or more of the following: audit the endpoint, clear all connections for the endpoint, or send a NotificationRequest asking the endpoint to process quarantined events (see 7.4.3.7).

A disconnected endpoint may wish to send a command (besides RestartInProgress) while it is disconnected. Doing so will only succeed once the Media Gateway Controller is reachable again, which raises the question of what to do with such a command meanwhile. At one extreme, the endpoint could drop the command right away; however, that would not work very well when the Media Gateway Controller was in fact available, but the endpoint had not yet completed the "disconnected" procedure (consider for example the case where a NotificationRequest was just received which immediately resulted in a Notify being generated). To prevent such scenarios, disconnected endpoints MUST NOT blindly drop new commands to be sent for a period of Ts_{max} seconds after they receive a non-audit command.

One way of satisfying this requirement is to employ a temporary buffering of commands to be sent, however in doing so, the endpoint must ensure that it:

– does not build up a long queue of commands to be sent;

– does not swamp the Media Gateway Controller by rapidly sending too many commands once it is connected again.

Buffering commands for Ts_{max} seconds and, once the endpoint is connected again, limiting the rate at which buffered commands are sent to one outstanding command per endpoint is considered safe. If the endpoint is not connected within Ts_{max} seconds, but a "disconnected" procedure is initiated within Ts_{max} seconds, the endpoint MAY piggy-back the buffered command(s) with that RestartInProgress. Note that once a command has been sent, regardless of whether it was buffered

initially, or piggy-backed earlier, retransmission of that command MUST cease Ts_{max} seconds after the initial send as described in 7.4.2.

The disconnected procedure is complete once a success response has been received. Error responses are handled similarly to the restart procedure (clause 7.4.3.5). If the "disconnected" procedure is to be initiated again following an error response, the rate-limiting timer considerations specified above still apply.

Also note, that if the RestartInProgress is piggy-backed with the response (R) to a command received while being disconnected, then retransmission of the RestartInProgress does not require piggy-backing of the response R. However, while the endpoint is disconnected, resending the response R does require the RestartInProgress to be piggy-backed as well to ensure the in-order delivery of the two.

Note that if a disconnected procedure is already in progress when a command is received, the existing disconnect procedure MUST be terminated and a new procedure MUST be started. This is to support a possible Media Gateway Controller redirection.

Note also that a disconnected endpoint does not mean that the endpoint is in an "out-of-service" state. "Disconnected" is not a state of the endpoint's service availability, but rather an indication of the gateway's inability to communicate with its MGC.

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.

7.4.3.7 Call agent handling of disconnected endpoints

When an endpoint is in the "disconnected" state, it may accumulate a large number of events in the quarantine buffer. Also, a "disconnected" endpoint may autonomously delete established connections (say the gateway reboots). Therefore, when connectivity between a "disconnected" endpoint and its Call Agent is subsequently restored, the Call Agent MUST be prepared to deal with the following issues:

- The large number of Notify messages that may be generated by the endpoint if all events on the quarantine list are processed.
- The reception of old/stale events reported by the endpoint that no longer have any relevance. The quarantine buffer is a First-In-First-Out (FIFO) queue, where the oldest events are processed first and notified (if requested) to the Call Agent. The action taken by the Call Agent on receiving an old event may not be meaningful if the old event has been superseded by newer events (for example, an "MT/sup" event would no longer be relevant if the endpoint subsequently went on-hook).
- Connection state mismatch between the Call Agent and endpoint, where the Call Agent thinks the endpoint has one or more connections, but the endpoint in fact has no connections.

Call Agents are free to use any mechanism supported by the protocol to deal with the above issues. One way of achieving this is to do the following:

1) We define a new boolean variable called "disconnect-event-sync", which is maintained by the Call Agent for each of its endpoints. When set to true, this variable indicates that connectivity with a "disconnected" endpoint has recently been restored, but that event/signal synchronization has not yet been achieved. (Note, this variable is being introduced in order to describe Call Agent behaviour, and is not intended to imply any particular implementation. The variable is not externally visible.)

a) As soon as the Call Agent learns that an endpoint is disconnected, it sets "disconnect-event-sync" to true. The "disconnected" procedure ensures that the Call Agent will learn about the endpoint being disconnected by receiving a "disconnected" RestartInProgress message. When an endpoint receives a positive acknowledgement to the "disconnected" RSIP, it completes the "disconnected" procedure. At this point, the endpoint could immediately generate a Notify for two reasons: to send a Notify command that was buffered while the endpoint was disconnected, or, if the endpoint was in the "notification" state and "loop" mode while disconnected, to report a Notify-triggering event on the quarantine list.

   - If the endpoint operates in "step" mode, responding to the Notify will in itself not enable any further Notify messages to be generated (an additional NotificationRequest would be needed for that).
   - However if the endpoint operates in "loop" mode, then a response to the Notify will enable further Notify messages to be generated. As discussed above, this is sometimes undesirable as the events being reported may be old and a lot of events may have been quarantined, which in turn will result in a large number of Notify messages and subsequent NotificationRequest messages based on stale information.

b) As long as an endpoint has "disconnect-event-sync" set to true, the Call Agent should take action to ensure that the potentially large number of events on the quarantine list are either discarded, or processed in a controlled and orderly manner. This can be achieved in a number of ways:

   - The Call Agent can send a single NotificationRequest specifying that all quarantined events are to be discarded. On receiving a positive acknowledgment to this command, or on receiving a Notify with the same RequestIdentifier, the Call Agent should set the "disconnect-event-sync" indicator to false, at which point regular event processing for the endpoint is resumed. The downside to this approach is that it will clear any events that have been accumulated, regardless of the number of events accumulated. In some cases, this may result in unnecessary service interruption. In order to address this, protocol extensions will be necessary.
   - The Call Agent can send a NotificationRequest specifying that quarantined events are to be processed. If the endpoint is operating in "step" mode then it will report a single notify-triggering event for each NotificationRequest received, whereas in the "loop" mode it can report multiple events under a single NotificationRequest.

     Since the information conveyed by the notified events may no longer be relevant, the Call Agent should not blindly process these events (for example, on being notified of an "MT/sup" origination event, the Call Agent should not automatically send a NotificationRequest to request notification of "MT/inf" digits). Instead, the Call Agent must synchronize its internal state data with the actual state of the endpoint. Since an endpoint performs signal/event glare processing against the current seizure state of the circuit, the Call Agent can discover the current seizure state based on the response to a NotificationRequest. For example, a "402 – circuit not seized" response to a NotificationRequest asking for detection of "MT/inf" implies that the MT trunk is currently not seized. The Call Agent may choose to ignore notified events that are incompatible with the current hook-switch state (ignore "MT/inf" report of digits if the trunk is not seized, for example).

     Once all events on the quarantine list have been processed, the Call Agent should set the "disconnect-event-sync" indicator to false. The Call Agent can safely assume that all quarantined events have been processed if a delay of duration Tt_{hist} has expired since the Call Agent last prompted the endpoint to process the next

event (i.e., if a delay of duration Tt_{hist} has expired since the Call Agent sent the response to the previous Notify in the "loop" mode, or since the Call Agent received the last positive response to NotificationRequest in the "step" mode).

2) When endpoints become disconnected, connections created on that endpoint should be unaffected. However, it is always possible, that a connection can no longer be sustained by the endpoint and hence that it is deleted; this will result in a DeleteConnection command sent to the Call Agent. When the endpoint is disconnected, such a command may never make it to the Call Agent, in which case it will not learn about the deleted connection. Consequently, whenever a Call Agent learns that an endpoint is disconnected, it should audit the endpoint for the list of connections present on it.

7.5 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 five value ranges have been defined:

- value 000 indicates a response acknowledgement[24];
- 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 Table 3:

Table 3/J.171.1 – Return codes

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 or circuit already seized
402	The phone is already on-hook or circuit not seized.
407	Transaction aborted. The transaction was aborted by some external action, e.g., a ModifyConnection command aborted by a DeleteConnection command.
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.
503	"All of" wildcard not fully supported. The transaction contained an "all of" wildcard; however, the gateway does not fully support these. Note that this is currently only permissible for non-empty NotificationRequests.

[24] Response acknowledgement is used for provisional responses (see 8.8).

Table 3/J.171.1 – Return codes

Code	Meaning
505	Unsupported RemoteConnectionDescriptor. This SHOULD be used when one or more mandatory parameters or values in the RemoteConnectionDescriptor is not supported.
506	Unable to satisfy both LocalConnectionOptions and RemoteConnectionDescriptor. This SHOULD be used when the LocalConnectionOptions and RemoteConnectionDescriptor contain one or more mandatory parameters or values that conflict with each other and/or cannot be supported at the same time (except for codec negotiation failure – see error code 534).
510	The transaction could not be executed because a protocol error was detected.
511	The transaction could not be executed because the command contained an unrecognized extension.
512	The transaction could not be executed because the gateway is not equipped to detect one of the requested events.
513	The transaction could not be executed because the gateway is not equipped to generate one of the requested signals.
514	The transaction could not be executed because the gateway cannot send the specified announcement.
515	The transaction refers to an incorrect connection-id (may have been already deleted).
516	The transaction refers to an unknown call-id.
517	Unsupported or invalid mode.
518	Unsupported or unknown package.
519	Endpoint does not have a digit map.
520	The transaction could not be executed because the endpoint is "restarting".
521	Endpoint redirected to another MGC.
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.
534	Codec negotiation failure.
538	Event/signal parameter error (e.g., parameter missing, erroneous, unsupported, unknown, etc.).

7.6 Reason codes

Reason codes are used by the gateway when deleting a connection to inform the MGC about the reason for deleting the connection. The reason code is an integer number, and the values have been defined in Table 4:

Table 4/J.171.1 – Reason codes

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

7.7 Use of local connection options and connection descriptors

The normal sequence in setting up a bidirectional connection involves at least three steps:

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

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

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

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

- LocalConnectionOptions: Supplied by the Media gateway controller/Call Agent to control the media parameters used by the gateway for the connection. When supplied, the gateway must conform to these media parameters until either the connection is deleted, or a ModifyConnection command is received.
- RemoteConnectionDescriptor: Supplied by the Media gateway controller/Call Agent to convey the media parameters supported by the other side of the connection. When supplied, the gateway must conform to these media parameters until either the connection is deleted, or a ModifyConnection command is received.
- LocalConnectionDescriptor: Supplied by the gateway to the Media gateway controller/Call Agent to convey the media parameters it supports for the connection. When supplied, the gateway must honor the media parameters until either the connection is deleted, or the gateway issues a new LocalConnectionDescriptor. In addition to the media parameters assigned to the connection, the gateway can also advertise additional supported capabilities in the LocalConnectionDescriptor. Note that such capabilities MUST be provided outside m = line in the SDP. The gateway is free to advertise all of its supported capabilities independent of the LCO or RCD parameters received from the Call Agent, and independent of the media parameters associated with the connection.

Codec and packetization period selection MUST only be performed, as described in this clause, if either:

a) The gateway receives a CRCX; or

b) The gateway receives a MDCX and any of the following parameters are present:
   - encoding method (a: in LocalConnectionOptions);
   - packetization period (p: in LocalConnectionOptions);

– multiple packetization period (mp: in LocalConnectionOptions);
– RemoteConnectionDescriptor.

Furthermore, this codec and packetization period selection process MUST only use the information present in the connection request and not retain any of the values that may have been received in previous connection requests. For example, if a gateway received a MDCX with all necessary LCO parameters but was missing a RemoteConnectionDescriptor, it will negotiate as if no RemoteConnectionDescriptor had ever been received for that connection. As well, if all of the above parameters are omitted in a MDCX command, the existing negotiated codecs and packetization periods will remain intact.

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

- A list of codecs and packetization periods provided in the LocalConnectionOptions. A codec is allowed by the LocalConnectionOptions if it satisfies the constraints specified by the encoding method, packetization period and multiple packetization periods fields. If one or more of these fields are omitted, the omitted fields do not impose any constraints on the allowed codecs.
- A list of codecs and packetization periods in the RemoteConnectionDescriptor.
- An internal list of codecs and packetization periods that the gateway can support for the connection. A gateway may support one or more codecs and packetization periods for a given connection.

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

1) An approved list of codecs/packetization periods is formed by taking the intersection of the internal list of codecs/packetization periods and codecs/packetization periods allowed by the LocalConnectionOptions. If LocalConnectionOptions was not provided, the approved list of codecs/packetization periods thus contains the internal list. If the LocalConnectionOptions was provided but the codecs parameter was omitted, the LocalConnectionOptions implicitly allows all codecs in the internal list, provided they are compatible with any packetization period(s) specified. Similarly, if the LocalConnectionOptions was provided but the packetization period(s) was omitted, the LocalConnectionOptions implicitly contains the set of packetization periods supported by the internal list.
2) If the approved list of codecs/packetization periods is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended).
3) Otherwise, a negotiated list of codecs/packetization periods is formed by taking the intersection of the approved list of codecs/packetization periods and codecs/packetization periods allowed by the RemoteConnectionDescriptor. If a RemoteConnectionDescriptor was not provided, the negotiated list of codecs/packetization periods thus contains the approved list of codecs/packetization periods. If the RemoteConnectionDescriptor does not contain any media stream lines, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended). If the RemoteConnectionDescriptor contains multiple media streams, the MG SHOULD only accept one of these and reject the others by setting their port to zero in the LocalConnectionDescriptor. If the RemoteConnectionDescriptor was provided but the packetization period(s) was omitted, the negotiated list of packetization periods contains the set of packetization periods from the approved list. The MG MUST choose reasonable

defaults (as per RFC 2327) if the packetization period is explicitly omitted from both the LocalConnectionOptions and the RemoteConnectionDescriptor.

4) If the negotiated list of codecs/packetization periods is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended).

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

Note that the packetization interval for T.38 is selected using the same procedure that is used for audio codecs, as described above.

In the case that a gateway supports more than one codec per endpoint, there are two options the gateway can use in deciding how many codecs it wants to support for that connection:

1) Gateway supports multiple codecs and can switch between different codecs in real-time. The gateway returns all negotiated codecs in the SDP media stream line. Multiple codecs in the m = line means the device must be ready to receive media packets from any of the negotiated codecs. In addition, the gateway may send media packets from any of the negotiated codecs and switch between them as required.

2) Gateway supports one or more codecs but cannot switch between different codecs in real-time. The gateway therefore negotiates and returns only one codec in the SDP media stream line (optionally, gateway also puts additional supported codecs in the SDP 'X-pc-codecs' attribute). With this method, a codec change must be initiated by the MGC in order to change codecs.

7.7.1 RFC 2833 negotiation

The internal list of supported codecs MUST include the telephone-event codec with events 0-15. This will ensure that RFC 2833 DTMF relay will be used for the connection when authorized by the LCO (via inclusion or an empty a: parameter) and allowed by the RCD.

An example of an LCO that authorizes RFC 2833 DTMF relay is:

```
L: a:PCMU;PCMA;telephone-event, mp:10;20;-
```

The telephone-event codec MUST NOT be the only codec supplied in the LCO from the MGC. If the endpoint receives an LCO containing only the telephone-event codec, then it MUST return error code 524 – internal inconsistency in LocalConnectionOptions. If the approved list of codecs, as described in 7.7, contains only the telephone-event codec, the endpoint MUST return error code 534 – codec negotiation failure. Likewise, if the negotiated list of codecs, as described in 7.7, contains only the telephone-event codec, the endpoint MUST return error code 534.

If the packetization period LCO parameter is used for the connection, the endpoint MUST use that packetization rate for the DTMF relay packets. If the multiple packetization period parameter is used in the LCO, the MGC MUST use a hyphen for the packetization rate for the telephone-event codec. If the endpoint receives an LCO with the multiple packetization period parameter with a packetization rate for telephone-event codec not set to a hyphen, the endpoint MUST return error code 524 – Inconsistent LCO. When an endpoint returns an LCD which includes the ability to receive the telephone-event codec, it MUST use a hyphen as the packetization rate in the mptime SDP attribute.

An example of an LCD which advertises support of RFC 2833 DTMF digits is:

```
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 8 105
a=mptime:10 20 -
a=rtpmap:105 telephone-event/8000/1
```

For more information on the use of RFC 2833 DTMF Digits, refer to IPCablecom Audio/Video Codec Specification J.161.

7.7.2 Remote IP and port negotiation

The remote IP and port are provided via the Remote Connection Descriptor. Once obtained, in a successful connection handling command (for example, a Modify Connection command), the endpoint MUST continue to use them until either a new Remote Connection Descriptor is provided which specifies a new remote IP or port, or the connection is deleted. Note that the receipt of a Modify Connection command without a Remote Connection Descriptor which may kick off Codec negotiation does not invalidate the current remote IP and port, even if the media is modified from audio to image or vice versa.

8 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 MGC to a gateway. The Notify command is sent by the gateway to the MGC. The gateway can also send a DeleteConnection as defined in 7.3.6. The MGC can send either of the Audit commands to the gateway and, finally, the gateway can send a RestartInProgress command to the MGC.

8.1 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.

8.2 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.

8.2.1 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. Trunking gateways SHOULD use exactly one ASCII space separator; however, they MUST be able to parse messages with additional white space characters.

8.2.1.1 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 Table 5:

Table 5/J.171.1 – Request verb codes

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

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

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

8.2.1.2 Transaction identifiers

Transaction identifiers are used to correlate commands and responses.

A trunking gateway supports two separate transaction identifier name spaces:

- a transaction identifier name space for sending transactions; and
- a transaction identifier name space for receiving transactions.

At a minimum, transaction identifiers for commands sent to a given trunking gateway MUST be unique for the maximum lifetime of the transactions within the collection of MGCs that control that trunking gateway (see 8.5). Thus, regardless of the sending MGC, trunking gateways can always detect duplicate transactions by simply examining the transaction identifier. The coordination of these transaction identifiers between MGCs is however outside the scope of this Recommendation.

Transaction identifiers for all commands sent from a given trunking gateway MUST be unique for the maximum lifetime of the transactions (see 8.5) regardless of which MGC the command is sent to. Thus, an MGC can always detect a duplicate transaction from a trunking gateway by the combination of the domain-name of the endpoint and the transaction identifier. The gateway 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. Transaction identifiers SHOULD NOT use any leading zeroes. Equality is based on numerical value and leading zeroes are ignored. An MGCP entity MUST NOT reuse a transaction identifier more quickly than three minutes after completion of the previous command in which the identifier was used.

8.2.1.3 Endpoint, Media Gateway Controller and NotifiedEntity name coding

The endpoint names and MGC names are encoded as e-mail addresses, as defined in IETF RFC 2821. 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:

ds/ds1-3/2@TGCP2.whatever.net	Second circuit on the third DS1 in the trunking gateway TGCP2 in the "whatever" network.
MGC@mgc.whatever.net	Media Gateway Controller 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:

```
MGC@mgc.whatever.net:5234
```

In case the port number is omitted, the default MGCP port (2727 unless provisioned otherwise) will be used. Additional detail on endpoint names can be found in 7.1.1.

8.2.1.4 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 "TGCP" 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 TGCP 1.0
```

The "TGCP 1.0" portion signals that this is the TGCP 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).

8.2.2 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 however still case insensitive. The parameters that can be present in commands are defined in Table 6:

Table 6/J.171.1 – Command parameters

Parameter name	Code	Parameter value
ResponseAck (Note)	K	See description.
CallId	C	Hexadecimal string, MUST not exceed 32 characters.
ConnectionId	I	Hexadecimal string, MUST not exceed 32 characters.
NotifiedEntity	N	An identifier, in IETF 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	The RequestIdentifier hexadecimal string, length MUST NOT exceed 32 characters.
LocalConnectionOptions	L	See description.
ConnectionMode	M	See description.
RequestedEvents	R	See description.
SignalRequests	S	See description.
ObservedEvents	O	See description.
ConnectionParameters	P	See description.
ReasonCode	E	See description.
SpecificEndPointId	Z	An identifier, in IETF RFC 821 format, composed of an arbitrary string, optionally followed by an "@" followed by the domain name of the trunking gateway to which this endpoint is attached.
MaxEndPointIds	ZM	Decimal string, MUST not exceed 16 characters.
NumEndPoints	ZN	Decimal string, MUST not exceed 16 characters.
RequestedInfo	F	See description.
QuarantineHandling	Q	See description.
DetectEvents	T	See description.
EventStates	ES	See description.
RestartMethod	RM	See description.
RestartDelay	RD	A number of seconds encoded as a decimal number.
Capabilities	A	See description.
VersionSupported	VS	See description.
MaxMGCPDatagram	MD	See description
NOTE – The ResponseAck parameter was not shown in 7.3 as transaction identifiers are not visible in our example API. Implementers may choose a different approach.		

The parameters are not necessarily present in all commands. Table 7 provides the association between parameters and commands. The letter M stands for mandatory, O for optional, and F for forbidden:

Table 7/J.171.1 – Parameter association with command request

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ResponseAck (Note)	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	O	O	F	F	F	F	F	F
ConnectionMode	M	O	F	F	F	F	F	F
RequestedEvents	O[a)]	O[a)]	O[a)]	O[a)]	F	F	F	F
SignalRequests	O[a)]	O[a)]	O[a)]	O[a)]	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
Connection parameters	F	F	O	F	F	F	F	F
SpecificEndpointId	F	F	F	F	F	O	F	F
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
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
MaxMGCPDatagram	F	F	F	F	F	F	F	F
RemoteConnectionDescriptor	O	O	F	F	F	F	F	F

a) 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, omission of these two parameters when the command includes a RequestIdentifier means the corresponding lists will be considered empty.

NOTE – The ResponseAck parameter was not shown in 7.3 as transaction identifiers are not visible in our example API. Implementers may choose a different approach.

Trunking gateways and MGCs SHOULD always provide mandatory parameters before optional ones; however, trunking gateways 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.

8.2.2.1 Response acknowledgement

The response acknowledgement parameter is used to support the three-way handshake described in 8.7. 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
```

8.2.2.2 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest (includes notification request embedded in connection handling primitives) that triggered it. A RequestIdentifier is a hexadecimal string; length MUST NOT exceed 32 characters. RequestIdentifiers are compared as strings rather than numerical values. The string "0" is reserved for reporting of persistent events in the case where no NotificationRequest has been received yet (see 7.3.2).

8.2.2.3 Local connection options

The local connection options describe the operational parameters that the MGCs instruct 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 multiple packetization period in milliseconds for each codec in the encoding method LCO, encoded as the keyword "mp" followed by a colon and a list of decimal numbers or hyphens, with one entry for each entry in the Encoding Method field. Each packetization period value is separated from its successor by a single semicolon. The first entry in the list MUST be a decimal number. Subsequent entries in the list MUST be either a decimal number or a hyphen.
- The literal name of the compression algorithm, as specified in ITU-T Rec. J.161, encoded as the keyword "a" followed by a colon and a character string. If the Media Gateway Controller (MGC) specifies a list of values, these values will be separated by a semicolon. For RTP, audio codecs MUST be specified by using encoding names defined in the RTP AV Profile RFC 1890 encoding names registered with the IANA, or encoding names referenced or defined in the IPCablecom Audio/Video Codecs Specification. Non-audio media registered as a MIME type MUST use the "<MIME type>/<MIME subtype>" form, as in "image/t38". It is RECOMMENDED that other well-known variants of the literal codec names be supported as well.
- 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 Security are encoded as follows:

- 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 MUST be separated by a single 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 MUST be separated by a single 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 IETF RFC 2234. White spaces MUST not be sent within a ciphersuite or between adjacent ciphersuites when multiple ciphersuites are provided. The following example illustrates the formatting of a ciphersuite and a ciphersuite list:

```
sc-rtp 62/51;64/51;60/50
```

The actual list of IPCablecom supported ciphersuites is provided in ITU-T Rec. J.170.

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:G729, 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).

8.2.2.4 Capabilities

Capabilities inform the MGC 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. The literal names defined in Table 3 of the IPCablecom Audio/Video Codecs Specification (J.161) MUST be used. A list of values may be specified in which case the values will be separated by a semicolon;
- the bandwidth in kilobits per second (1000 bits per second), encoded as the keyword "b" followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen;
- the echo-cancellation parameter, encoded as the keyword "e" followed by a colon and the value "on" if echo cancellation is supported; "off" otherwise;
- the type of service parameter, encoded as the keyword "t" followed by a colon and the value "0" if type of service is not supported, all other values indicated support for type of service;
- the silence suppression parameter, encoded as the keyword "s" followed by a colon and the value "on" if silence suppression is supported; "off" otherwise;
- the event packages supported by this endpoint encoded as the keyword "v" followed by a colon and then a semicolon-separated list of package names supported. The first value specified will be the default package for the endpoint;
- the connection modes supported by this endpoint encoded as the keyword "m" followed by a colon and a semicolon-separated list of connection modes supported as defined in 8.2.2.7;
- the keyword "sc-rtp" followed by a colon and a semicolon-separated list of RTP ciphersuites, using the same encoding as in the LocalConnectionOptions;
- the keyword "sc-rtcp" followed by a colon and a semicolon-separated list of RTCP ciphersuites, using the same encoding as in the LocalConnectionOptions.

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

Examples of capabilities are:

```
A: a:PCMU; p:10-30, e:on, s:off, v:IT,
              m:sendonly;recvonly;sendrecv;inactive
A: a:G729; p:30-90, e:on, s:on, v:IT,
              m:sendonly;recvonly;sendrecv;inactive,
              sc-rtp: 64/51;60/51, sc-rtcp:71/81
```

Note that the codecs and security algorithms are merely examples – separate IPCablecom Recommendations detail the actual codecs and algorithms supported, as well as the encoding used (see ITU-T Recs J.170, J.162 and J.161). 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 (see RFC 1827 and RFC 1034). Note also that each set of capabilities is provided on a single line. The examples above show each set on multiple lines due only to formatting restraints of this Recommendation.

8.2.2.5 Connection parameters

Connection parameters are encoded as a string of type and value pairs, where the type is one of the codes in Table 8 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 Table 8:

Table 8/J.171.1 – Connection parameter types

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.
Remote Packets sent	PC/RPS	The number of packets that were sent on the connection from the perspective of the remote endpoint.
Remote Octets sent	PC/ROS	The number of octets that were sent on the connection from the perspective of the remote endpoint.
Remote Packets lost	PC/RPL	The number of packets that were not received on the connection, as deduced from gaps in the sequence number from the perspective of the remote endpoint.
Remote Jitter	PC/RJI	The average inter-packet arrival jitter, in milliseconds, expressed as an integer number from the perspective of the remote endpoint.

Extension connection parameters names are composed of the string "X-" followed by a two-or three-letter extension parameter name. MGCs that receive unrecognized extensions MUST silently ignore these extensions. If an endpoint receives RTCP packets with these statistics, it MUST return the Remote parameters (Rxx above) in the response to the Delete Connection and Audit Connection commands.

An example of a connection parameter encoding is:

```
P: PS=1245, OS=62345, PR=0, OR=0, PL=0, JI=0, LA=48, PC/RPS=0, PC/ROS=0,
PC/RPL=0, PC/RJI=0
```

8.2.2.6 Reason codes

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

```
E: 900 Endpoint malfunctioning
```

A list of reason codes can be found in 7.6.

8.2.2.7 Connection mode

The connection mode describes the connection's operation mode. The possible values are indicated in Table 9:

Table 9/J.171.1 – Connection mode values

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: inactive	The gateway should neither send nor receive packets
M: loopback	The gateway should place the endpoint in Loopback mode
M: conttest	The gateway should place the endpoint in Continuity Test 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

8.2.2.8 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 "co", "Co", "CO" or "cO" should be considered equal.

The following are valid examples of event names:

`IT/co1`	Originating continuity test in the ISUP trunk package
`MT/oc`	Operation Complete in the MF Terminating Protocol package
`co1`	Originating continuity test in the ISUP trunk package, assuming that the ISUP trunk package is the default package for the endpoint
`IT/rt@0A3F58`	Ringback on connection "0A3F58"

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

`IT/all`	All events in the ISUP trunk 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 valid examples of such notations:

`IT/ma@*`	The RTP media start event on all connections for the endpoint
`IT/rt@$`	Ringback on the current connection

An initial set of event packages for trunking gateways can be found in Annex A.

8.2.2.9 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 7.3.1 for valid combinations. The actions, when specified, are encoded as a list of keywords enclosed in parentheses and separated by commas. The codes for the various actions are indicated in Table 10:

Table 10/J.171.1 – Action codes

Action	Code
Notify immediately	N
Accumulate	A
Ignore	I
Keep signal(s) active	K
Embedded NotificationRequest	E
Embedded ModifyConnection	C

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

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

`R: oc(N), of(N)` Notify operation complete, notify operation failure.

The embedded NotificationRequest follows the format:

`E ( R( <RequestedEvents>), S( <SignalRequests>) )`

with each of R, and S being optional and possibly supplied in another order.

The embedded ModifyConnection action follows the format:

`C(M(<ConnectionMode`$_1$`>( <ConnectionID`$_1$`> )) , ... ,`
`M(<ConnectionMode`$_n$`>(ConnectionID`$_n$` )))`

The following example illustrates the use of embedded ModifyConnection:

`R: ma@23B34D(A, C(M(sendrecv($)))), oc(N), of(N)`

On media start on connection "23B34D", change the connection mode of the "current connection" to "send receive". Notify events on "operation complete" and "operation failure".

8.2.2.10 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. The MG MUST support, at a minimum, a single signal on each endpoint and simultaneously support the generation of one signal on each connection for a given endpoint. Specific packages MAY define requirements beyond these minimum capabilities. For signal combinations beyond this minimum requirement that the MG does not support, it SHOULD return error code 502.

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 7.3.1):

- 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 "mysignal" signal on:

```
mysignal(+), mysignal
```

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 ringback tone signal for 6 seconds:

```
rt(to=6000)
rt(to(6000))
```

Individual signals may define additional signal parameters.

The signal parameters will be enclosed within parentheses, as in the following hypothetical example:

```
S: display(10/14/17/26, "555 1212", CableLabs)
```

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

```
S: signal1, signal2
```

8.2.2.11 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 are:

```
O: ma@A43B81
O: ft
O: IT/ft
O: IT/ft, IT/mt
```

8.2.2.12 RequestedInfo

The RequestedInfo parameter contains a comma-separated list of parameter codes, as defined in 8.2.2 "Parameter lines"; clause 7.3.8 lists the parameters that can be audited. The following values in Table 11 are supported as well:

Table 11/J.171.1 – RequestedInfo values

RequestedInfo parameter	Code
LocalConnectionDescriptor	LC
RemoteConnectionDescriptor	RC

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

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

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

```
F: A
```

8.2.2.13 QuarantineHandling

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

– The keyword "process" or "discard" to indicate the treatment of quarantined observed events. If neither process nor discard is present, process is assumed.

– The keyword "step" or "loop" to indicate whether, at most, one notification is expected, or whether multiple notifications are allowed. If neither "step" nor "loop" is present, "step" is assumed. Support for these two keywords is mandatory.

The following values are valid examples:

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

8.2.2.14 DetectEvents

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

```
T: ft, mt
```

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

8.2.2.15 EventStates

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

```
ES: MO/rlc
```

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

8.2.2.16 RestartMethod

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

```
RM: restart
```

8.2.2.17 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 TGCP 1.0
```

8.2.2.18 Call Identifier

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

8.2.2.19 Connection Identifier

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

8.2.2.20 MaxMGCPDatagram

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

```
MD: 8100
```

8.3 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
```

Table 12 summarizes the response parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response. 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.

Table 12/J.171.1 – Parameter association with command response

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ResponseAck (Note 5)	O (Note 1)	O (Note 1)	O (Note 1)	O (Note 1)	O (Note 1)	O (Note 1)	O (Note 1)	O (Note 1)
CallId	F	F	F	F	F	F	O	F
ConnectionId	O (Note 2)	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
ConnectionMode	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
ConnectionParameters	F	F	O (Note 3)	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
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

Table 12/J.171.1 – Parameter association with command response

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
LocalConnection Descriptor	O (Note 4)	O (Note 4)	F	F	F	F	O	F
MaxMGCPDatagram	F	F	F	F	F	O	F	F
RemoteConnection Descriptor	F	F	F	F	F	F	O	F

NOTE 1 – 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.

NOTE 2 – In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter and a LocalConnectionDescriptor. It may also be followed by a Specific-Endpoint-Id parameter, if the creation request was sent to a wildcarded Endpoint-Id. The Connection-Id and LocalConnectionDescriptor parameter are marked as optional in the table. In fact, they are mandatory with all positive responses, when a connection was created, and forbidden when the response is negative, and no connection was created.

NOTE 3 – Connection-Parameters are only valid in a successful response to a non-wildcarded DeleteConnection command sent by the Call Agent.

NOTE 4 – A LocalConnectionDescriptor MUST be transmitted with a positive response (code 200) to a CreateConnection. It MUST also be transmitted in response to a ModifyConnection command, if the modification resulted in a modification of the session parameters. The LocalConnectionDescriptor is encoded as a "session description", as defined in 8.4. It is separated from the response header by an empty line.

NOTE 5 – The ResponseAck parameter was not shown in 7.3 as transaction identifiers are not visible in our example API. Implementers may choose a different approach.

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

8.3.1 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 8.4. 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 18 96 97 0
a=rtpmap:96 G726-32/8000
a=rtpmap:97 telephone-event/8000
a=mptime:20 10 – 10
```

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

```
200 1204 OK
K:
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 18 96 97 0
a=rtpmap:96 G726-32/8000
a=rtpmap:97 telephone-event/8000
a=mptime:20 10 – 10
```

The final response is acknowledged by a Response Acknowledgement:

```
000 1204
```

8.3.2 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 8.4. 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
a=mptime: 20
```

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
```

8.3.3 DeleteConnection

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in 8.2.2.5.

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

8.3.4 NotificationRequest

A NotificationRequest response does not include any additional response parameters.

8.3.5 Notify

A Notify response does not include any additional response parameters.

8.3.6 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 will still be provided. Each local endpoint name "expanded" by a wild-card character will appear on a separate line using the "SpecificEndPointId" parameter code, e.g.:

```
200 1200 OK
Z: ds/ds1-1/1@tgw.whatever.net
Z: ds/ds1-1/2@tgw.whatever.net
ZN: 24
```

An example of a response to an AuditEndPoint message containing a non-wildcarded endpoint name is shown below. Note that the SpecificEndPointId is not provided in this case. Note also that each set of capabilities is provided on a single line. The example below shows each set on multiple lines due only to formatting restraints of this Recommendation.

```
200 1200 OK
A: a:PCMU; p:10, e:on, s:off, t:1, v:IT,
     m:sendonly;recvonly;sendrecv;inactive
A: a:G728, p:20, e:on, s:off, t:1, v:IT,
     m:sendonly;recvonly;sendrecv;inactive
A: a:G729A; p:30-90, e:on, s:on, t:1, v:L,
     m:sendonly;recvonly;sendrecv;inactive;confrnce
```

8.3.7 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: mp:20;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 G728/8000
a=mptime: 10
```

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.

8.3.8 RestartInProgress

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

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

8.4 Session description encoding

The session description is encoded in conformance with the session description protocol (SDP); however, trunking gateways 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 IETF 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 "image", the session description is for an image service such as FAX.

8.4.1 SDP audio service use

In a trunking gateway we only have to describe sessions that use exactly one MIME media type at a given point in time; either "audio" (for voice or voiceband data) or "image" (for T.38 fax calls). The parameters of SDP that are relevant for both the "audio" and "image" based media types are specified in 8.4.2. Parameters specified to "audio" are specified in 8.4.3. Parameters specific to "image" when used for T.38 are specified in 8.4.4. Trunking gateway MUST support session descriptions that conform to these rules and in the following order:

1) the SDP profile presented below;

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

The MGC should be careful if it is necessary to alter the SDP received from an endpoint. SDP provides a means of communicating the capabilities of an endpoint to another endpoint. If the MGC chooses to modify the SDP, it MUST NOT alter the SDP such that it violates the rules defined in this clause.

The SDP profile provided describes the use of the session description protocol in TGCP. The general description and explanation of the individual parameters as well as most of the parameters being used for audio only can be found in IETF RFC 2327; parameters specific to T.38 for image can be found in T.38. Below we detail what values TGCP endpoints need to provide for these fields (send) and what TGCP endpoints should do with values supplied or not supplied for these fields (receive). It should be noted that the SDP profile used here does not comply with the offer/answer model defined in RFC 3264. Thus, if a MGC needs to interact with another entity that uses the offer/answer model, the MGC may need to edit the SDP it receives from the endpoint.

8.4.2 SDP parameters common to both audio and image service use

8.4.2.1 Protocol Version (v=)

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

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

Receive: MUST be provided in accordance with IETF RFC 2327.

8.4.2.2 Origin (o=)

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

```
o=<username> <session-ID> <version> <network-type> <address-type> <address>
o=-  2987933615    2987933615 IN IP4       126.16.64.4
```

Username:

Send: Hyphen MUST be used as username when privacy is requested. Hyphen SHOULD be used otherwise.[25] Since TGCP endpoints do not know when privacy is requested, they SHOULD always use a hyphen.

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

Receive: This field SHOULD be ignored.

Session-ID:

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

Receive: This field SHOULD be ignored.

Version:

Send: In accordance with IETF 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 IETF RFC 2327 for interoperability with non-IPCablecom clients.

Receive: This field MUST be ignored.

8.4.2.3 Session name (s=)

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

Send: Hyphen MUST be used as Session name.

Receive: This field MUST be ignored.

8.4.2.4 Session and media information (i=)

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

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

Receive: This field MUST be ignored.

8.4.2.5 URI (u=)

```
u= <URI>
```

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

Receive: This field MUST be ignored.

8.4.2.6 E-Mail address and phone number (e=, p=)

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

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

Receive: This field MUST be ignored.

8.4.2.7 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.

8.4.2.8 Bandwidth (b=)

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

Send: Bandwidth information is optional in SDP but it SHOULD always be included[26]. When an rtpmap or a non well-known codec[27] 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. Refer to the IPCablecom Codec specification (J.161) clause 7.5 for the details of calculating the bandwidth value.

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

---

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

[27] A non well-known codec is a codec not defined in the IPCablecom codec (ITU-T Rec. J.161).

8.4.2.9 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 IETF RFC 2327.

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

8.4.2.10 Attributes (a=)

```
a= <attribute> : <value>
a= mptime: <alternative 1> <alternative 2> …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.

Receive: One or more of the "a" attribute lines specified below MAY be included and MUST be acted upon accordingly.

Note that SDP requires unknown attributes to be ignored.

mptime: This attribute defines a list of packetization period values the endpoint is capable of using (sending and receiving) for this connection.

Send: This attribute MUST be present. There MUST be precisely one entry in the list for each <format> entry provided in the "m=" line. Entry number j in this list defines the packetization period for entry number j in the "m=" line. The first entry in the list MUST be a decimal number whereas subsequent entries in the list MUST be either a decimal number or a hyphen. For those media formats where a single packetization rate does not apply (e.g., non-voice codecs such as telephone event or comfort noise), a hyphen ("-") MUST be encoded at the corresponding location in the list of packetization periods.

Receive: Conveys the list of packetization periods that the remote endpoint is capable of using for this connection; one for each media format in the "m=" line. For media formats whose packetization period is specified as a hyphen ("-"), the endpoint MUST use one of the packetization periods that was actually specified in the list. If the "mptime" attribute is absent, then the value of the "ptime" attribute, if present, MUST be taken as indicating the packetization period for all codecs present in the "m=" line. If neither the "mptime" nor the "ptime" attribute is present, then the MG must assume the default value for well-known codecs (as defined in RFC 1890).

X-pc-bridge:

Send: TGCP endpoints MUST NOT use this attribute.

Receive: TGCP endpoints MUST ignore this attribute if received.

recvonly:

Send: This field should not be supplied by a TGCP endpoint.

Receive: The field MUST be ignored.

sendrecv:

Send: This field should not be supplied by a TGCP endpoint.

Receive: The field MUST be ignored.

sendonly:

Send: This field should not be supplied by a TGCP endpoint.

Receive: The field MUST be ignored.

ptime:

Send: The ptime SHOULD be sent if it was received in a RemoteConnectionDescriptor or if the MGC used the packetization period ('p:') LocalConnectionOptions.

Receive: The field MUST be ignored if the SDP contains the "mptime" attribute (as required in PacketCable compliant devices). If the "mptime" attribute is not present, then this field is used to define the packetization interval for all codecs present in the SDP description. If neither "mptime" nor "ptime" is present, the MG MUST assume reasonable default values for well-known codecs as defined in ITU-T Rec. J.161.

8.4.3 SDP audio service use

The following SDP parameters are applied at the media level and are specific to Audio Service Use. PacketCable endpoints MUST NOT send any of these parameters within an image media descriptor (see 8.4.4). If, however, the endpoint receives an SDP with attribute parameters specific to image only within an audio media descriptor, the parameters SHOULD be ignored. Further, when media capability parameters are to be provided, each media capability descriptor (which includes the capability description line, a=cdsc, along with 0 or more capability attribute lines, e.g., a=cpar) MUST appear after the last media attribute and each media capability descriptor MUST be listed separately.

8.4.3.1 Encryption keys

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

Security services for IPCablecom are defined by ITU-T Rec. J.170. The security services specified for RTP and RTCP do not comply with those of IETF RFC 1889, IETF RFC 1890, and IETF 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.

8.4.3.2 Attributes (a=)

```
a=<attribute> : <value>
a=rtpmap : <payload type> <encoding name>/<clock rate> [/<encoding parameters>]
a=rtpmap : 0              PCMU / 8000
a=fmtp:<format> <format specific parameters>
a=X-pc-codecs: <alternative 1> <alternative 2> ...
a=X-pc-secret: <method>:<encryption key>[pad]
a =X-pc-csuites-rtp: <alternative 1> <alternative 2> ...
a =X-pc-csuites-rtcp: <alternative 1> <alternative 2> ...
```

Send: One or more of the "a" attribute lines specified below MAY be included.

Receive: One or more of the "a" attribute lines specified below MAY be included and MUST be acted upon accordingly.

rtpmap:

Send: When used, the field MUST be used in accordance with IETF RFC 2327. It MAY be used for well-known as well as non well-known codecs. The encoding names used are provided in a separate IPCablecom Recommendation (see ITU-T Recs J.161 and J.170). On a given connection, the dynamic payload type for a given encoding method MUST be the same in the send and receive direction. Furthermore, on a given connection, once a dynamic payload type has been mapped to a given encoding method, that payload type MUST NOT subsequently be mapped to another encoding method.

Receive: When used, the field MUST be used in accordance with IETF RFC 2327. For a given connection, implementations SHOULD NOT fail if a given encoding method is mapped to different payload types in the send and receive direction, or if a given payload type is remapped.

fmtp:

Send: This field MAY be used to provide parameters specific to a particular format. For example, the field could be used to describe telephone events supported for an RFC 2833 format. When used, the format MUST be one of the formats specified for the media. The parameters specified are provided in a separate specification that details the usage of the format.

Receive: When used, 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 and (possibly) the PAD, to be used for RTP and RTCP security. The secret and PAD are encoded similarly to the encryption key (k=) parameter of IETF 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 IETF 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.

The requirements for when to transmit PAD are described in the J.170 security Recommendation. If present, it MUST be separated by at least one whitespace from the secret. PAD and secret MUST use the same encode method.

Receive: Conveys the end-to-end secret to be used for RTP and RTCP security. If present, its use is as described in the Security Recommendation (J.170) and it MUST be separated by at least one white space from the secret. PAD and secret MUST use the same encode method.

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 IETF RFC 2234. Whitespaces are not allowed within a ciphersuite. The following example illustrates the use of ciphersuite:

```
62/51
```

The actual list of ciphersuites is provided in ITU-T Rec. J.170.

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.

8.4.3.3 Media Announcements (m=)

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

```
M=<media> <port>   <transport>   <format>[<format>]
M=audio        3456      RTP/AVP        0    97
```

Media:

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

Receive: The type received MUST be "audio".

Port:

Send: MUST be filled in accordance with IETF 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 IETF 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 IETF RFC 2327 MUST be used. Specifically, this field contains a list of one or more RTP payload types that this endpoint is prepared to receive on the connection and that it would prefer to send with. Each payload type is mapped uniquely to a codec, either statically or dynamically. The static mapping SHOULD be used if available (e.g., 0 for PCMU, 8 for PCMA). If a dynamic payload mapping is used, an RTPMAP attribute MUST also be present and the guidelines in 8.4.3.2 MUST be followed.

Receive: In accordance with IETF RFC 2327. Specifically this indicates the payload type(s) that the other side of this connection is prepared to receive.

8.4.4 SDP image service use for T.38

The following SDP parameters are applied at the media level and are specific to Image Service Use for T.38. IPCablecom endpoints MUST NOT send any of these parameters within an audio media descriptor (see 8.4.3). If, however, the endpoint receives an SDP with attribute parameters specific to audio only within an image media descriptor, the parameters SHOULD be ignored. Further, when media capability parameters are to be provided, each media capability descriptor (which includes the capability description line, a=cdsc, along with 0 or more capability attribute lines, e.g., a=cpar) MUST appear after the last media attribute and each media capability descriptor MUST be listed separately.

8.4.4.1 Encryption Keys

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

There are currently no security services defined for the "image/t38" media type.

Send: MUST NOT be used.

Receive: This field SHOULD be ignored.

8.4.4.2 Attributes (a=)

```
a= <attribute> : <value>

a=T38FaxVersion: <version>

a=T38MaxBitrate: <bitrate>

a=T38FaxRateManagement: <faxratemanagement>

a=T38FaxMaxBuffer: <maxbuffer>

a=T38FaxMaxDatagram: <maxsize>

a=T38FaxUdpEC: <ECmethod>

a=T38FaxFillBitRemoval

a=T38FaxTranscodingMMR

a=T38FaxTranscodingJBIG
```

Send: One or more of the "a" attribute lines specified below MAY be included.

Receive: One or more of the "a" attribute lines specified below MAY be included and MUST be acted upon accordingly. Attribute values are case-insensitive. Implementations MUST accept the lowercase, uppercase, and mixed upper/lowercase encodings of all attributes.

Note that SDP requires unknown attributes to be ignored.

T38FaxVersion:

As defined in T.38: The recipient of the offer MUST accept that version or modify the version attribute to be an equal or lower version when transmitting an answer to the initial offer. The recipient of an offer MUST NOT respond with an answer containing a higher version than that which was offered.

Also as defined in T.38: Early implementations of T.38 equipment may not provide a T.38 version number. In receipt of SDP without the version attribute, the endpoint MUST assume that the version is 0. This is applied in the following discussion on sending and receiving this attribute:

Send: The endpoint MUST indicate the version that it intends to use with the T38FaxVersion attribute. However, it MUST NOT indicate a version that is higher than the version received in a RemoteConnectionDescriptor.

Receive: If a RemoteConnectionDescriptor is received and the T38FaxVersion attribute is not included, then the endpoint MUST use version 0 of the T.38 specification. If the attribute is included, the endpoint MUST use a version of the specification that is the same or lower than the version indicated.

T38MaxBitRate:

Send: The T38MaxBitRate attribute MUST NOT be included.

Receive: The T38MaxBitRate attribute SHOULD be ignored.

T38FaxRateManagement:

Send: The T38FaxRateManagement attribute MUST be included and MUST have a value of "transferredTCF" when UDPTL is used. With the value "transferredTCF", TCF is passed end-to-end as opposed to an attribute value of "localTCF" where TCF is generated locally. Note that "localTCF" is only appropriate when a reliable transport such as TCP is used.

Receive: When UDPTL is used, the T38FaxRateManagement attribute either MUST be present with a value of "transferredTCF" or it MUST be absent, in which case transferred TCF is assumed. All other values of the attribute MUST be rejected (error code 505 – Unsupported RemoteConnectionDescriptor).

T38FaxMaxBuffer:

Send: The T38FaxMaxBuffer attribute MUST NOT be included.

Receive: The T38FaxMaxBuffer attribute SHOULD be ignored.

T38FaxMaxDatagram:

Send: The T38FaxMaxDatagram attribute MUST be included. The value indicated MUST NOT be less than 160 bytes. This is based on 40 ms packetization period and a 14,400 bit/s data rate. It includes the UDPTL datagram without the IP and UDP headers.

Receive: Endpoints MUST NOT send a datagram larger than that specified in the T38FaxMaxDatagram attribute. Prior to sending any T.38 datagram, the endpoint MUST ensure that is within the limits defined by this attribute. If the specified T38FaxMaxDatagram value is too small to support redundancy for a given datagram, but sufficient to support T.38 without redundancy, then the endpoint MUST send that T.38 datagram without redundancy. If the value is too small to allow the datagram to be sent without redundancy, the endpoint MUST NOT send the T.38 datagram and the endpoint MUST then generate a failure indication.

T38FaxUdpEC:

Support for redundancy is mandatory whereas support for forward error correction is optional. Use of either scheme requires negotiation.

Send: The T38FaxUdpEC attribute MUST be included. The value "t38UDPFEC" MAY be sent if FEC is supported and there either was no RCD provided with the command or the value of the attribute received in the RCD for this command is "t38UDPFEC". Otherwise "t38UDPRedundancy" MUST be sent.

Receive: Redundancy MUST be used if the value of the T38FaxUdpEC attribute is "t38UDPRedundancy". If the T38FaxUdpEC attribute is "t38UDPFEC" and FEC is supported by the endpoint, then FEC SHOULD be used. If the T38FaxUdpEC attribute is "t38UDPFEC" and FEC is not supported, then redundancy MUST be used. If this attribute is not included, the endpoint MUST NOT use redundancy or FEC.

T38FaxFillBitRemoval:

Support for fill bit removal is optional and any use of it needs to be negotiated.

Send: If fill bit insertion and removal is supported and desired to be used, and the command either did not include an RCD, or it included an RCD with the T38FaxFillBitRemoval attribute present, then T38FaxFillBitRemoval MUST be included and fill bit insertion and removal MUST then be used. In all other cases, the T38FaxFillBitRemoval attribute MUST NOT be included and fill bit insertion and removal MUST NOT be used.

Receive: Fill bit insertion and removal MUST NOT be used if the T38FaxFillBitRemoval attribute is absent.

T38FaxTranscodingMMR:

MMR transcoding does not apply to UDPTL-based T.38.

Send: When UDPTL is being used for T.38, the T38FaxTranscodingMMR attribute MUST NOT be included.

Receive: If the T38FaxTranscodingMMR attribute is present for UDPTL-based T.38, the command MUST be rejected (error code 505 – unsupported RemoteConnectionDescriptor).

T38FaxTranscodingJBIG:

JBIG transcoding does not apply to UDPTL-based T.38.

Send: When UDPTL is being used for T.38, the T38FaxTranscodingJBIG attribute MUST NOT be included.

Receive: If the T38FaxTranscodingJBIG attribute is present for UDPTL-based T.38, the command MUST be rejected (error code 505 – unsupported RemoteConnectionDescriptor).

8.4.4.3 Media Announcements (m=)

Media Announcements (m=) consists of 4 sub-fields:

```
m= <media> <port> <transport> <fmt list>
"m= image 3456 udptl t38"
```

Media:

Send: The 'image' media type MUST be used for UDPTL-based T.38.

Receive: The type received MUST be 'image' for UDPTL-based T.38.

Port:

Send: MUST be filled in accordance with RFC 2327. The port specified is the receive port. 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 'udptl' MUST be used for UDPTL-based T.38.

Receive: The transport protocol MUST be 'udptl' for UDPTL-based T.38. Implementations SHOULD also tolerate the upper-case form "UDPTL", as well as mixed upper and lower-case forms of the string "udptl".

Media Formats:

Send: The media format MUST be "t38".

Receive: The media format MUST be "t38".

8.5 Transmission over UDP

8.5.1 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 MGC. 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[28], the commands MUST be sent to the default MGCP port, which is 2427 for Commands sent to Gateways and 2727 for commands sent to Media Gateway Controllers. To minimize backward compatibility issues, it is RECOMMENDED that the Media Gateway Controller always explicitly states the MGCP port to use in TGCP messages (and not rely on the default).

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 time-outs have been exceeded. A retransmission strategy is specified in 8.5.2.

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 7.4.

---

[28] Each endpoint may be provisioned with a separate MGC address and port.

8.5.2 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. Trunking gateways MUST implement a retransmission strategy using exponential back-off with configurable initial and maximum retransmission timer values.

Trunking gateways 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.

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.

8.5.3 Maximum datagram size, fragmentation and reassembly

TGCP messages being transmitted over UDP rely on IP for fragmentation and reassembly of large datagrams. The maximum theoretical size of an IP datagram is 65 535 bytes. With a 20-byte IP header and an 8-byte UDP header, this leaves us with a maximum theoretical TGCP message size of 65 507 bytes when using UDP.

However, IP does not require a host to receive IP datagrams larger than 576 bytes (RFC 1122), which would provide an unacceptably small TGCP message size. Consequently, TGCP mandates that implementations MUST support TGCP datagrams up to at least 4000 bytes, which requires the corresponding IP fragmentation and reassembly to be supported. Note that the 4000-byte limit applies to the TGCP level. Lower layer overhead will require support for IP datagrams that are larger than this: UDP and IP overhead will be at least 28 bytes, and, e.g., use of IPSec will add additional overhead.

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

8.6 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 contains a single dot, as in for example:

```
200 2005 OK
.
DLCX 1210 ds/ds1-1/1@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
```

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

Piggybacking can be used to achieve two purposes:

- Guaranteed in-order delivery and processing of messages.
- Fate sharing of message delivery.

When piggybacking is used to guarantee in-order delivery of messages, entities MUST ensure that this in-order delivery property is retained on retransmissions of the individual messages. An example of this is when multiple Notify's are sent using piggy backing (as described in 7.4.3.1).

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

8.7 Transaction identifiers and three-way handshake

Transaction identifiers are integer numbers in the range from 1 to 999 999 999. MGCs 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. MGCs 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 MGC 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[29] is received, the response that is being acknowledged by it can be

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

deleted. Gateways should silently discard further commands from that MGC 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 wild-cards:
   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" wild-card, and the endpoint-name returned in the response was $term_{new}$.
   c) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wild-card, and $term_{new}$ is covered by the wild-card in $term_{old}$.
   d) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "any-of" wild-card, no endpoint-name was returned, and $term_{new}$ is covered by the wild-card in $term_{old}$.
2) If $term_{new}$ contains the "all" wild-card:
   a) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wild-card, and $term_{new}$ is covered by the wild-card in $term_{old}$.
3) If $term_{new}$ contains the "any of" wild-card:
   a) Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wild-card, and $term_{new}$ is covered by the wild-card in $term_{old}$ if the "any of" wild-card in $term_{new}$ was replaced with the "all" wild-card.

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

The following examples illustrate the use of these rules:

- If $term_{new}$ is "ds/ds1-2/1" and $term_{old}$ is "ds/ds1-2/1", then the old response can be confirmed per rule 1a.
- If $term_{new}$ is "ds/ds1-1/3" and $term_{old}$ is "*", then the old response can be confirmed per rule 1c.
- If $term_{new}$ is "ds/ds1-2/*" and $term_{old}$ is "*", then the old response can be confirmed per rule 2a.
- If $term_{new}$ is "ds/ds1-2/$" and $term_{old}$ is "ds/ds1-2/*", 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 MGC, 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.

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.

8.8 Provisional responses

In some cases, transaction completion times may be significantly longer than otherwise. TGCP 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 a small period of time (200 ms is RECOMMENDED). 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 (error code 407 SHOULD be used).

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 7.4), i.e., the sender of the final response will retransmit it if the "Response Acknowledgement" is not received in time. The "Response Acknowledgment " response is never acknowledged.

9 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 TGCP are provided in ITU-T Rec. J.170, 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 trunking gateways.

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 MGC. 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 MGC. Otherwise, the following symbols identify the type of event:
  - "**OO**" On/Off signal. The signal is turned on until commanded by the MGC 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.

A.1 ISUP trunk package

Package name: IT.

The following codes are used to identify events and signals for the "IT" package. An MG that supports the "IT" package MUST support all events and signals listed here.

Table A.1/J.171.1 – ISUP trunk package events and signals

Code	Description	Event	Signal	Additional information
co1	Continuity tone 1	√	TO	Time-out = 3 seconds
co2	Continuity tone 2	√	TO	Time-out = 3 seconds
ft	Fax tone	√	–	
ld	Long duration connection	C	–	
ma	Media start	C	–	
mt	Modem tone	√	–	
oc	Operation complete	√	–	
of	Operation failure	√	–	
ro	Reorder tone	–	TO	Time-out = 30 seconds
rt	Ringback tone	–	C, TO	Time-out = 180 seconds
TDD	Telecommunications Device for the Deaf (TDD) tones	√		

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

Continuity Tone 1 (co1): A tone at 2010 Hz per ITU-T Rec. Q.724. To conform with current continuity testing practice, the event SHOULD NOT be generated until the tone has been removed. The tone is of type TO – the continuity test will only be applied for the specified period of time. The provisioning process may alter the default value.

Continuity Tone 2 (co2): A tone at 1780 Hz per ITU-T Rec. Q.724. To conform with current continuity testing practice, the event SHOULD NOT be generated until the tone has been removed. The tone is of type TO – the continuity test will only be applied for the specified period of time. The provisioning process may alter the default value.

The continuity tones are used when the MGC wants to initiate a continuity test. There are two types of tests: single tone and dual tone. The party originating the continuity check signals and detects the appropriate tones for the trunk in question. For instance, for a continuity test from a 4-wire to a 2-wire circuit the following messages could be used:

Originating gateway

```
RQNT 1234 ds/ds3-1/ds1-6/17@tgw1.example.net
X: AB123FE0
S: co2
R: co1
```

Terminating gateway

```
CRCX 1234 ds/ds1-4/7@tgw2.example.net
C: A3C47F21456789F0
L: p:10, a:PCMU
M: conttest
```

The originating gateway sends the requested signal, and looks for the return of the appropriate tone for the trunk in question. When it detects that tone and deems the continuity check to be successful, it generates the "co1" event which in the example will be notified to the MGC. If the test does not succeed prior to the time-out, an "operation complete" event will be generated and in this case sent to the MGC. Similarly, if an error occurs prior to the time-out, an "operation failure" event will be generated. The "oc" and "of" events will be parameterized with the name of the event/signal they report, i.e., "co1" in this case.

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

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.

Media start (ma): The media start event occurs on a connection when the first valid[30] 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 communication is detected – see e.g., ITU-T Rec. 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 "continuity tone 1". 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: IT/oc(IT/co1)
```

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

```
O: IT/oc(IT/rt@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: IT/oc(B/C)
```

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

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: IT/of(IT/co2)
```

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

```
O: IT/of(IT/rt@0A3F58)
```

When the operation failure event is requested, event parameters cannot 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: IT/of(B/C(M(sendrecv(AB2354))))
```

Reorder tone (ro): Reorder tone, a.k.a congestion tone, is specified in ITU-T Rec. E.180/Q.35.

Ring back tone (rt): Audible Ring Tone is specified in ITU-T Rec. E.180/Q.35. The definition of the tone is defined by the national characteristics of the Ringback Tone, and MAY be established via provisioning. The ringback signal can be applied to both an endpoint and a connection.

Telecommunications Device for the Deaf (TDD) tones: The TDD event is generated whenever a TDD communication is detected – see e.g., ITU-T Rec. V.18.

Appendix I

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 circuit endpoint). 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 circuit endpoint plays the incoming stream over the circuit and generates the outgoing stream from the circuit 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 endpoint.

Table I.1 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".

Table I.1/J.171.1 – Mixing of different connections when one or more connections are concurrently active

		Connection A mode					
		sendonly	**recvonly**	**sendrecv**	**loopback/ conttest**	**inactive**	**netwloop/ netwtest**
Connection B mode	**sendonly**	$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}= A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}=cot$	$A_{out}= NA$ $B_{out}= H_{in}$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$
	recvonly		$A_{out}= NA$ $B_{out}= NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}= H_{in}$ $B_{out}= NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}=cot$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}= B_{in}$	$A_{out}= A_{in}$ $B_{out}= NA$ $H_{out}= B_{in}$
	sendrecv			$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}=cot$	$A_{out}= NA$ $B_{out}= H_{in}$ $H_{out}= B_{in}$	$A_{out}= A_{in}$ $B_{out}= H_{in}$ $H_{out}= B_{in}$
	loopback/ conttest				$A_{out}=NA$ $B_{out}=NA$ $H_{out}=cot$	$A_{out}= NA$ $B_{out}= NA$ $H_{out}= cot$	$A_{out}= NA$ $B_{out}= NA$ $H_{out}= cot$
	inactive					$A_{out}= NA$ $B_{out}= NA$ $H_{out}= NA$	$A_{out}= A_{in}$ $B_{out}= NA$ $H_{out}=NA$
	netwloop/ netwtest						$A_{out}= A_{in}$ $B_{out}= B_{in}$ $H_{out}=NA$

Connections in "network loopback", "network continuity test", or "inactive" modes are not affected by connections in the "active" modes. Table I.1 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 Trunk;
- 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 endpoint, where "cot" indicates continuity test, whether in the "continuity test" or "loopback" mode;
- NA indicates No Stream whatever.

Appendix II

Example command encodings

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

II.1 NotificationRequest

The first example illustrates a NotificationRequest that will initiate a continuity test and look for the verification of the test. The "notified entity" for the endpoint will be set to "ca@ca1.whatever.net:5678" and the RequestIdentifier will be repeated in the corresponding Notify command:

```
RQNT 1201 ds/ds1-1/2@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
N: mgc@mgc1.whatever.net:5678
X: 0123456789AC
R: co1, oc(N), of(N)
S: co1
```

The response indicates that the transaction was successful:

```
200 1201 OK
```

II.2 Notify

The example below illustrates a Notify message that notifies a successful continuity test as indicated by the observed events. Since a "notified entity" was specified in the triggering NotificationRequest, it is repeated here. Also, the RequestIdentifier is included to correlate this Notify command with the NotificationRequest (includes notification request embedded in connection handling primitives) command that triggered:

```
NTFY 2002 ds/ds1-1/2@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
N: mgc@mgc1.whatever.net:5678
X: 0123456789AC
O: co1
```

The Notify response indicates that the transaction was successful:

```
200 2002 OK
```

II.3 CreateConnection

The first example illustrates a CreateConnection command to create a connection on the endpoint specified. The connection will be part of the specified CallId. The LocalConnectionOptions specify that G.711 μ-law will be the codec used and the packetization period will be 10 ms. The connection mode will be "receive only":

```
CRCX 1204 ds/ds1-1/17@tgw2.whatever.net MGCP 1.0 TGCP 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
a=mptime:10
```

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

```
CRCX 1205 ds/ds1-1/1@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
X: 0123456789AD
R: MO/sup(addr(K0, 4,1,1, s2), id(K0,0,0,7,3,2,5,5,5,1,2,3,4,s0))
S: MO/ans

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=mptime:10
```

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

```
401 2005 Circuit already seized
```

Our third example illustrates the use of the provisional response and the three-way handshake:

```
CRCX 1206 ds/ds1-1/1@tgw.whatever.net MGCP 1.0 TGCP 1.0
K: 1205
C: A3C47F21456789F0
L: p:10, a:PCMU
M: inactive

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0 18
a=mptime:10
```

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=mptime:10
```

Note that the endpoint elected to support only the PCMU codec, i.e., payload number 0.

A little later, the final response is received:

```
200 1206 OK
K:
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The MGC acknowledges the final response as requested:

```
000 1206
```

and the transaction is complete.

II.4 ModifyConnection

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

```
MDCX 1209 ds/ds1-1/21@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
N: mgc@mgc1.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 PSTN until it detects audio on the connection specified for the ring-back signal:

```
MDCX 1210 ds/ds1-1/3@abc5.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: ma@ FDE234C8
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
a=mptime:10
```

The response indicates that the transaction was successful:

```
200 1206 OK
```

II.5 DeleteConnection (from the Media Gateway Controller)

In this example, the MGC simply instructs the trunking gateway to delete the connection FDE234C8 on the endpoint specified:

```
DLCX 1210 ds/ds1-1/1@tgw.whatever.net MGCP 1.0 TGCP 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
```

II.6 DeleteConnection (from the trunking gateway)

In this example, the trunking gateway sends a DeleteConnection command to the MGC 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 ds/ds1-1/1@tgw-2567.whatever.net MGCP 1.0 TGCP 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 MGC sends a success response to the gateway:

```
200 1210 OK
```

II.7 DeleteConnection (multiple connections from the Media Gateway Controller)

In the first example, the MGC instructs the trunking gateway to delete all connections related to call `"A3C47F21456789F0"` on the specified endpoint:

```
DLCX 1210 ds/ds1-1/6@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
```

The response indicates success and that the connection(s) were deleted:

```
250 1210 OK
```

In the second example, the MGC instructs the trunking gateway to delete all connections related to all of the endpoints specified:

```
DLCX 1210 ds/ds1-1/*@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
```

The response indicates success:

```
250 1210 OK
```

II.8 AuditEndpoint

In the first example, the MGC wants to learn what endpoints are present on the trunking gateway specified, hence the use of the "all of" wild-card for the local portion of the endpoint-name. The MGC only wants two endpoint names:

```
AUEP 1200 *@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
ZM: 2
```

The trunking gateway indicates success and includes a list of up to two endpoint names. A total of 24 endpoint names matched the wild-card specified:

```
200 1200 OK
Z: ds/ds1-1/1@tgw-2567.whatever.net
Z: ds/ds1-1/2@tgw-2567.whatever.net
ZN: 24
```

In the second example, the capabilities of one of the endpoints is requested:

```
AUEP 1201 ds/ds1-1/1@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
F: A
```

The response indicates success and the capabilities as well. Two codecs are supported, but with different capabilities. Consequently, two separate capability sets are returned:

```
200 1201 OK
A: a:PCMU, p:10-100, e:on, s:off, v:IT, m:sendonly;recvonly;sendrecv;
                    inactive;loopback;conttest;netwloop;netwtest
A: a:G728, p:30-90, e:on, s:on, v:IT, m: sendonly;recvonly;sendrecv;
                         inactive;loopback;conttest;netwloop
```

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

```
AUEP 2002 ds/ds1-1/1@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
F: R, S,X,N,I,T,O,ES
```

The response indicates success:

```
200 2002 OK
R: IT/ft,mt(N)
S:
X: 0123456789B1
N: [128.96.41.12]
I: 32F345E2
T: ft
O:
ES:
```

The list of requested events contains two 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 "IT/ft" event. The omission of a value for the "SignalRequests" means there are

currently no active signals. The current "notified entity" refers to an IP-address and only a single connection exists for the endpoint. The current value of DetectEvents is "ft", and the list of ObservedEvents is empty as is the EventStates.

II.9 AuditConnection

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

```
AUCX 2003 ds/ds1-1/18@tgw-2567.whatever.net MGCP 1.0 TGCP 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: mgc@mgc1.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
a=mptime:10
```

In the second example, there is a request to audit RemoteConnectionDescriptor and LocalConnectionDescriptor:

```
AUCX 1203 ds/ds1-1/2@tgw.whatever.net MGCP 1.0 TGCP 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
a=mptime:10

v=0
```

II.10 RestartInProgress

The first example illustrates a RestartInProgress message sent by a trunking gateway to inform the MGC that the specified endpoint will be taken out of service in 300 seconds:

```
RSIP 1200 ds/ds1-1/1@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
RM: graceful
RD: 300
```

The MGCs response indicates that the transaction was successful:

```
200 1200 OK
```

In the second example, the RestartInProgress message sent by the trunking gateway informs the MGC that all of the trunking gateway'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 *@tgw-2567.whatever.net MGCP 1.0 TGCP 1.0
RM: restart
RD: 0
```

The MGCs response indicates success, and furthermore provides the endpoints in question with a new "notified entity":

```
200 1204 OK
N: MGC-1@whatever.net
```

Alternatively, the command could have failed with a new "notified entity" as in:

```
521 1204 OK
N: MGC-1@whatever.net
```

In that case, the command would then have to be retried in order to satisfy the "restart procedure" (see 7.4.3.5), this time going to MGC "`MGC-1@whatever.net`".

Appendix III

Example Call Flow

In this appendix, we provide an example call flow between an on-net user using an unspecified MTA and signalling protocol[31] and an off-net user accessed through a trunking gateway using the TGCP protocol and a signalling gateway supporting SS7 ISUP signalling. It should be noted that this call flow, although a valid one, is merely an example that may or may not be used in practice.

In the call flow below in Figure III.1, CMS refers to the Call Management Server, MGC refers to the Media Gateway Controller, TGW refers to the trunking gateway, and SG refers to the signalling gateway:

MTA	CMS	MGC	TGW	SG
Place call (E.164)	→			
	Call Setup	→		
		Create Connection(SDP1) + Notification Request	→	
		←	**Ack(SDP2)**	
		IAM	———	→
		←	**Notify**	
		Ack +	→	
		Modify Connection +		
		Notification Request		
		COT	———	→
		←	———	**ACM**
	←	***Alerting***		
←	***Alerting***	←	———	**ANM**
		ModifyConnection	→	
		←	**Ack**	
	←	***Answer***		
←	***Answer***			
		(Call Established)		
Hang up	→			
	Release	→		
		REL	———	→
		Delete Connection	→	
		←	**Ack(Perf data)**	
		←	———	**RLC**
		(Call end)		

Figure III.1/J.171.1 – Example call flow

During these exchanges, the TGCP profile of MGCP is used by the MGC to control the trunking gateway. An unspecified protocol is assumed between the MTA, CMS and MGC.

[31] This could be either NCS or DCS.

We assume that the MTA indicates (directly or indirectly) to the MGC its desire to establish a voice communication to an E.164 telephone number and that it includes a session description with this request. The CMS looks up the requested E.164 number and determines that it needs to place an off-net and therefore contacts the appropriate MGC. The MGC decides that it needs to place the call through trunking gateway `tgw.whatever.net`. Furthermore, the MGC decides that continuity testing should be performed for this call.

The first command is a combined CreateConnection and NotificationRequest command sent to the trunking gateway:

```
CRCX 2001 ds/ds1-1/6@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: inactive
X: 0123456789B0
R: co2, oc, of
S: co1

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=mptime:10
```

The trunking gateway, at that point, is instructed to start the continuity test and to look for the outcome of the test and report it. The generation of the continuity test signal and the detection of its success (or failure) via the event mechanism are synchronized, so when the "co2" event occurs, the "co1" test will stop. Note that the endpoint elected to support only the PCMU codec, i.e., payload number 0. The create connection portion of the command instructs to create an inactive connection on the endpoint specified using G.711 with a packetization period of 10 ms. Also, the command includes the session description received from the originating MTA.

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

```
200 2001 OK
I: 32F345E2

v=0
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
a=mptime:10
```

The MGC sends an SS7 IAM message through the signalling gateway to the switch connected to the trunk the call is being placed on. Included in this message is an indication that continuity testing is to be performed.

Subsequently, we assume the continuity test succeeds resulting in the "co2" event being generated and notified to the MGC:

```
NTFY 3001 ds/ds1-1/6@tgw.whatever.net MGCP 1.0 TGCP 1.0
X: 0123456789B0
O: co2
```

The MGC sends an SS7 COT indicating "continuity check successful" to the remote switch and Ack's the Notify command received. It also piggybacks a combined ModifyConnection and NotificationRequest command instructing the gateway to place the connection in "receive only" mode and to start looking for fax and modem tones:

```
200 3001 OK
.
MDCX 2006 ds/ds1-1/6@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: 32F345E2
M: recvonly
X: 0123456789B0
R: ft,mt
```

At this stage, the MGC has established a half-duplex transmission path. The phone attached to the ingress MTA 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.

The MGC then receives an SS7 ACM message indicating the called party is being alerted, and subsequently an SS7 ANM message indicating the called party has answered. The MGC places the connection in full-duplex mode by sending the following ModifyConnection command to the trunking gateway:

```
MDCX 2007 ds/ds1-1/6@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: 32F345E2
M: sendrecv
```

The trunking gateway immediately responds to the command:

```
200 2007 OK
```

In parallel, the MGC informs the originating MTA about the call answer event and records the call answer time.

At this point, the call is fully established.

At some later point in time, the phone attached to the originating MTA, in our scenario, goes on-hook and a hang-up event is relayed to the MGC (either directly or indirectly through the CMS as shown here) instructing the MGC that the call should end.

The MGC verifies that the call in fact should be disconnected, e.g., there is no facility-hold, and it therefore sends an SS7 REL message to the remote switch, and also a DeleteConnection command to the trunking gateway:

```
DLCX 2009 ds/ds1-1/6@tgw.whatever.net MGCP 1.0 TGCP 1.0
C: A3C47F21456789F0
I: 32F345E2
```

The trunking gateways will respond with an acknowledgement that includes the connection parameters for the connection:

```
250 2009 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

A confirmation to the call tear-down in the form of an SS7 RLC message is also received by the MGC which finally records the end of the call.

Appendix IV

Endpoint requirements

This appendix defines a set of TGCP endpoint specific requirements.

IV.1 Connection modes supported

Table IV.1 lists the connection modes that a given TGCP endpoint MUST support:

Table IV.1/J.171.1 – List of connection modes that must be supported by a TGCP endpoint

Endpoint type	Additional endpoint info	sendonly	recvonly	sendrecv	inactive	loopback	conttest	netwloop	netwtest
DS-0	ISUP trunk	√	√	√	√	√	√	√	√
DS-0	MF trunk	√	√	√	√	–	–	√	√

Appendix V

Compatibility information

This appendix provides TGCP protocol compatibility information.

V.1 NCS compatibility

This version of TGCP is based on and aligned with ITU-T Rec. J.162 as much as possible. Since TGCP and NCS address different types of gateways, a couple of differences do exist. The following summarizes these differences:

- **Connection modes**: NCS and TGCP share a common set of connection modes, but each also has a set of connection modes the other does not support:
  - NCS supports the connection modes "conference" and "replicate" which TGCP does not;
  - TGCP supports the connection modes "continuity test" and "loopback" which NCS does not.
- **Digit maps**: TGCP does not support digit maps; however, NCS does. This has a couple of implications, e.g.:
  - there is no TGCP command that can accept a digit map as a parameter;
  - the "Accumulate according to digit map" action is not supported in TGCP;
  - the "digit map" cannot be audited.
- **Dynamic Quality of Service**: NCS supports IPCablecom Dynamic of Quality of Service signalling, however TGCP does not.

In addition to the above, the following non-protocol-related differences exist between NCS and TGCP:

- **Event packages**: The initial event packages in TGCP and NCS are different.
- **Endpoint naming scheme**: The endpoint naming scheme for TGCP and NCS endpoints are slightly different.

V.2 MGCP compatibility

TGCP (and NCS) is furthermore a profile of IETF RFC 2705 MGCP 1.0; however, TGCP has introduced a couple of additions as well. The following lists TGCP additions that are currently not included in MGCP:

- **Endpoint naming scheme**: A specific endpoint naming scheme has been introduced for DS-0 endpoints. The rules for wild-carding are more restrictive than in MGCP, and also introduces the "range" concept for DS-0 endpoints.
- **Embedded ModifyConnection**: A new Embedded ModifyConnection action has been introduced.
- **Security**: IPCablecom Security services are supported in TGCP. 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 wild-card 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 MGCs and gateways to determine which protocol versions each supports.
- **Error codes**: Two new error codes have been introduced: 532 and 533.
- **Usage of SDP**: A new SDP usage profile is included in TGCP. 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 recommendation of the provisional response mechanism has been included in TGCP. 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 parentheses within signal parameters. All Time-Out signals can have their time-out value altered by a signal parameter.
- **Event packages**: TGCP introduces a set of new event packages.

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

Appendix VI

ABNF grammar for TGCP

RFC 3435 includes a formal description of the MGCP protocol syntax following the "Augmented BNF for Syntax Specifications". This formal description is referenced by developers for the creation of interoperable devices. A copy of the MGCP protocol syntax, annotated and edited to indicate its applicability to IPCablecom Recommendations, is provided in this appendix.

Implementations SHOULD conform to the portions of this ABNF grammar that relate to their respective specifications, i.e., NCS, TGCP. Also note that there are a few parameter encodings (e.g., embedded request, digit maps, vendor extension names) where the NCS grammar and/or TGCP grammar differ from the MGCP grammar.

Five annotations are used to distinguish between the following five different cases:

1) The language of the RFC has been changed to accommodate NCS and TGCP requirements.
2) The language of the RFC is applicable to NCS and possibly MGCP, not to TGCP.
3) The language of the RFC is applicable to TGCP and possibly MGCP, not to NCS.
4) The language of the RFC is only applicable to NCS and TGCP.
5) The language of the RFC is only applicable to MGCP.

The language in each case is indicated by a different type face as specified below.

;RFC 3435 grammar changed to accommodate NCS and TGCP

```
;Bold indicates NCS only (and possibly MGCP)
;Italics indicates TGCP only (and possibly MGCP)
;Bold italics indicates NCS and TGCP only
;Text in grey is for MGCP only
```

```
MGCPMessage = MGCPCommand / MGCPResponse
MGCPCommand = MGCPCommandLine 0*(MGCPParameter) [EOL *SDPinformation]
MGCPCommandLine = MGCPVerb 1*(WSP) transaction-id 1*(WSP)
                        endpointName 1*(WSP) MGCPversion EOL
MGCPVerb = "EPCF" / "CRCX" / "MDCX" / "DLCX" / "RQNT"
         / "NTFY" / "AUEP" / "AUCX" / "RSIP" / extensionVerb
extensionVerb  = ALPHA 3(ALPHA / DIGIT) ; experimental starts with X
transaction-id = 1*9(DIGIT)
endpointName       = LocalEndpointName "@" DomainName
LocalEndpointName = LocalNamePart 0*("/" LocalNamePart)
LocalNamePart      = AnyName / AllName / NameString
AnyName            = "$"
AllName            = "*"
NameString         = 1*(range-of-allowed-characters)
; VCHAR except "$", "*", "/", "@"
range-of-allowed-characters  = %x21-23 / %x25-29 / %x2B-2E
                             / %x30-3F / %x41-7E
DomainName = 1*255(ALPHA / DIGIT / "." / "-")    ; as defined
           / "#" number                          ; in RFC 821
           / "[" IPv4address / IPv6address "]"   ; see RFC 2373
; Rewritten to ABNF from RFC 821
number =  1*DIGIT
;From RFC 2373
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
; this production, while occurring in RFC 2373, is not referenced
; IPv6prefix  = hexpart "/" 1*2DIGIT
hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq  = hex4 *( ":" hex4)
hex4    = 1*4HEXDIG
MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
                            [1*(WSP) ProfileName]
ProfileName   = "NCS 1.0"    ; For NCS
              / "TGCP 1.0"   ; For TGCP
               / VCHAR *( WSP / VCHAR)
MGCPParameter = ParameterValue EOL
; Check infoCode if more parameter values defined
; Most optional values can only be omitted when auditing
ParameterValue = ("K"  ":" 0*(WSP)  [ResponseAck])
               / ("B"  ":" 0*(WSP)  [BearerInformation])
               / ("C"  ":" 0*(WSP)  CallId)
               / ("I"  ":" 0*(WSP)  [ConnectionId])
               / ("N"  ":" 0*(WSP)  [NotifiedEntity])
               / ("X"  ":" 0*(WSP)  [RequestIdentifier])
               / ("L"  ":" 0*(WSP)  [LocalConnectionOptions])
               / ("M"  ":" 0*(WSP)  ConnectionMode)
               / ("R"  ":" 0*(WSP)  [RequestedEvents])
               / ("S"  ":" 0*(WSP)  [SignalRequests])
               / ("D"  ":" 0*(WSP)  [DigitMap])  ; For NCS (and MGCP)
               / ("O"  ":" 0*(WSP)  [ObservedEvents])
               / ("P"  ":" 0*(WSP)  [ConnectionParameters])
               / ("E"  ":" 0*(WSP)  ReasonCode)
               / ("Z"  ":" 0*(WSP)  [SpecificEndpointID])
               / ("Z2" ":" 0*(WSP)  SecondEndpointID)
               / ("I2" ":" 0*(WSP)  SecondConnectionID)
               / ("F"  ":" 0*(WSP)  [RequestedInfo])
               / ("Q"  ":" 0*(WSP)  QuarantineHandling)
               / ("T"  ":" 0*(WSP)  [DetectEvents])
               / ("RM" ":" 0*(WSP)  RestartMethod)
               / ("RD" ":" 0*(WSP)  RestartDelay)
               / ("A"  ":" 0*(WSP)  [Capabilities])
               / ("ES" ":" 0*(WSP)  [EventStates])
               / ("PL" ":" 0*(WSP)  [PackageList])     ; Auditing only
```

```
                / ("MD" ":" 0*(WSP)  MaxMGCPDatagram)   ; Auditing only
                / (extensionParameter ":" 0*(WSP) [parameterString])
                / VersionSupported  ; NCS and TGCP - response only
                / MaxEndpointIds    ; NCS and TGCP
                / NumEndpoints      ; NCS and TGCP - response only
; <extensionParameter> ":" parameterString defined by NCS and TGCP
VersionSupported = "VS" ":" MGCPversion *( "," 0*(WSP) MGCPversion)
MaxEndpointIds  = "ZM" ":" 0*(WSP) 1*16(DIGIT)
NumEndpoints    = "ZN" ":" 0*(WSP) 1*16(DIGIT)  ; Responses only
; A final response may include an empty ResponseAck
ResponseAck =  confirmedTransactionIdRange
               *( ","  0*(WSP) confirmedTransactionIdRange )
confirmedTransactionIdRange = transaction-id ["-" transaction-id]
BearerInformation = BearerAttribute 0*("," 0*(WSP) BearerAttribute)
BearerAttribute   = ("e" ":" BearerEncoding)
                  / (BearerExtensionName [":" BearerExtensionValue])
BearerExtensionName  = PackageLCOExtensionName
BearerExtensionValue = LocalOptionExtensionValue
BearerEncoding = "A" / "mu"
CallId = 1*32(HEXDIG)
; The audit request response may include a list of identifiers
ConnectionId = 1*32(HEXDIG) 0*("," 0*(WSP) 1*32(HEXDIG))
SecondConnectionID = ConnectionId
NotifiedEntity = [LocalName "@"] DomainName [":" portNumber]
LocalName  = LocalEndpointName            ; No internal structure
portNumber = 1*5(DIGIT)
RequestIdentifier = 1*32(HEXDIG)
LocalConnectionOptions = LocalOptionValue 0*(WSP)
                         0*("," 0*(WSP) LocalOptionValue 0*(WSP))
LocalOptionValue = ("p"  ":" packetizationPeriod)
                 / ("a"  ":" compressionAlgorithm)
                 / ("b"  ":" bandwidth) ; Only for capabilities in
                                        ; NCS and TGCP
                 / ("e"  ":" echoCancellation)
                 / ("gc" ":" gainControl)
                 / ("s"  ":" silenceSuppression)
                 / ("t"  ":" typeOfService)
                 / ("r"  ":" resourceReservation)
                 / ("k"  ":" encryptiondata)
                 / ("nt" ":" ( typeOfNetwork /
                                supportedTypeOfNetwork))
                 / (LocalOptionExtensionName
                       [":" LocalOptionExtensionValue])
                 / MPacketizationPeriod   ; NCS and TGCP only
                 / RTPciphersuite         ; NCS and TGCP only
                 / RTCPciphersuite        ; NCS and TGCP only
                 / DQoSGateID             ; NCS only
                 / DQoSReservation        ; NCS only
                 / DQoSResourceID         ; NCS only
                 / DQoSReserveDestination ; NCS only
                 / CallContentId          ; TGCP only
                 / CallContentDestination ; TGCP only

Capabilities    =  CapabilityValue 0*(WSP)
                     0*("," 0*(WSP) CapabilityValue 0*(WSP))
CapabilityValue = LocalOptionValue
                / ("v" ":" supportedPackages)
                / ("m" ":" supportedModes)

PackageList     = pkgNameAndVers 0*("," pkgNameAndVers)
pkgNameAndVers  = packageName ":" packageVersion
packageVersion  = 1*(DIGIT)
; For NCS and TGCP, range format is only allowed for capabilities
; and not for LocalConnectionOptions.
```

```
packetizationPeriod  = 1*4(DIGIT) ["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
algorithmName        = 1*(SuitableLCOCharacter)
bandwidth            = 1*4(DIGIT) ["-" 1*4(DIGIT)]
echoCancellation     = "on" / "off"
gainControl          = "auto" / ["-"] 1*4(DIGIT)
silenceSuppression   = "on" / "off"
typeOfService        = 1*2(HEXDIG)     ; 1 hex only for capabilities
resourceReservation  = "g" / "cl" / "be"
;encryption parameters are coded as in SDP (RFC 2327)
;NOTE: encryption key may contain an algorithm as specified in RFC 1890
encryptiondata = ( "clear" ":" encryptionKey )
               / ( "base64" ":" encodedEncryptionKey )
               / ( "uri" ":" URItoObtainKey )
               / ( "prompt" ) ; defined in SDP, not usable in MGCP!
encryptionKey = 1*(SuitableLCOCharacter) / quotedString
; See RFC 2045
encodedEncryptionKey = 1*(ALPHA / DIGIT / "+" / "/" / "=")
URItoObtainKey = 1*(SuitableLCOCharacter) / quotedString
typeOfNetwork = "IN" / "ATM" / "LOCAL" / OtherTypeOfNetwork
; Registered with IANA - see RFC 2327
OtherTypeOfNetwork     = 1*(SuitableLCOCharacter)
supportedTypeOfNetwork = typeOfNetwork *(";" typeOfNetwork)
supportedModes    = ConnectionMode 0*(";" ConnectionMode)
supportedPackages = packageName 0*(";" packageName)
packageName = 1*(ALPHA / DIGIT / HYPHEN) ; Hyphen neither first or last
LocalOptionExtensionName = VendorLCOExtensionName
                         / PackageLCOExtensionName
                         / OtherLCOExtensionName
VendorLCOExtensionName   = "x" ("+"/"-") 1*32(SuitableExtLCOCharacter)
PackageLCOExtensionName  = packageName "/"
                           1*32(SuitablePkgExtLCOCharacter)
; must not start with "x-" or "x+"
OtherLCOExtensionName    = 1*32(SuitableExtLCOCharacter)
; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by NCS/TGCP
MPacketizationPeriod = "mp" ":" multiplepacketizationPeriod
multiplepacketizationPeriod = mpPeriod 0*(";" mpPeriod)
mpPeriod             = 1*4(DIGIT) / HYPHEN
RTPciphersuite   = "sc-rtp"  ":" ciphersuite
RTCPciphersuite  = "sc-rtcp" ":" ciphersuite
ciphersuite      = [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]
AuthenticationAlgorithm = 1*( ALPHA / DIGIT / "-" / "_" )
EncryptionAlgorithm     = 1*( ALPHA / DIGIT / "-" / "_" )
; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by NCS only
DQoSGateID             = "dq-gi" [":" 1*8(HEXDIG)]  ; Only empty for
                                                    ; capabilities
DQoSReservation        = "dq-rr" ":" DQoSResMode *(";" DQoSResMode)
DQoSResMode            = "sendresv" / "recvresv" / "snrcresv" /
                         "sendcomt" / "recvcomt" / "snrccomt"
DQoSResourceID         = "dq-ri" ":" 1*8(HEXDIG)
DQoSReserveDestination = "dq-rd" ":" IPv4address [":" portNumber]
; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by TGCP only
CallContentId          = "es-cci" ":" 1*8(HEXDIG)
CallContentDestination = "es-ccd" ":" IPv4address ":" portNumber

LocalOptionExtensionValue = (1*(SuitableExtLCOValChar)
                                                / quotedString)
                            *(";" (1*(SuitableExtLCOValChar)
                                                / quotedString))
```

```
;Note: No "data" mode.
ConnectionMode = "sendonly" / "recvonly" / "sendrecv"
               / "confrnce" / "inactive"
               / "loopback" / "conttest" ; TGCP (and MGCP) only
               / "replcate" ; NCS only
               / "netwloop" / "netwtest"
               / ExtensionConnectionMode
ExtensionConnectionMode = PkgExtConnectionMode
PkgExtConnectionMode    = packageName "/" 1*(ALPHA / DIGIT)
RequestedEvents = requestedEvent 0*("," 0*(WSP) requestedEvent)
requestedEvent  = (eventName ["(" requestedActions ")"])
                / (eventName "(" requestedActions ")"
                             "(" eventParameters ")" )
eventName = [(packageName / "*") "/"]
                (eventId / "all" / eventRange
                                       / "*" / "#") ; for DTMF
                                 ["@" (ConnectionId / "$" / "*")]
eventId = 1*(ALPHA / DIGIT / HYPHEN)   ; Hyphen neither first nor last
eventRange = "[" 1*(DigitMapLetter / (DIGIT "-" DIGIT) /
                         (DTMFLetter "-" DTMFLetter)) "]"
DTMFLetter = "A" / "B" / "C" / "D"
requestedActions = requestedAction 0*("," 0*(WSP) requestedAction)
requestedAction  = "N" / "A"
                 / "D" ; For NCS (and MGCP)
                 / "S" / "I" / "K"
                 / "E" "(" EmbeddedRequest ")"
                 / ExtensionAction
                 / "C" "(" EmbeddedModeChange     ; For NCS and TGCP
                     0*("," 0*WSP EmbeddedModeChange) ")"   ; only
; NCS and TGCP define the Embedded ModifyConnection action.
; MGCP grammar does not allow for the format used in NCS and TGCP:
EmbeddedModeChange = "M" "(" ConnectionMode "(" EmConnectionId ")" ")"
EmConnectionId     = ConnectionId / "$"
ExtensionAction  = PackageExtAction
PackageExtAction = packageName "/" Action ["(" ActionParameters ")"]
Action           = 1*ALPHA
ActionParameters = eventParameters        ; May contain actions
;NOTE: Should tolerate different order when receiving, e.g. for NCS
EmbeddedRequest = (      "R" "(" EmbeddedRequestList ")"
                    ["," 0*(WSP) "S" "(" EmbeddedSignalRequest ")"]
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"]      )
                / (      "S" "(" EmbeddedSignalRequest ")"
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"] )
                / (      "D" "(" EmbeddedDigitMap ")" )
                / NCSTGCPEmbeddedRequest
;Text below is for NCS and TGCP only. The difference compared to MGCP
;is simply that the order of the items is not fixed. Also for TGCP Digit Maps
; are not used
NCSTGCPEmbeddedRequest = NCSTGCPEmbeddedRequestItem
                         *2("," 0*(WSP) NCSTGCPEmbeddedRequestItem)
NCSTGCPEmbeddedRequestItem  = ("R" "(" EmbeddedRequestList ")" )
                            / ("S" "(" EmbeddedSignalRequest ")" )
                            / ("D" "(" EmbeddedDigitMap ")" )
EmbeddedRequestList   = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap
SignalRequests   = SignalRequest 0*("," 0*(WSP) SignalRequest )
SignalRequest    = eventName [ "(" eventParameters ")" ]
eventParameters  = eventParameter 0*("," 0*(WSP) eventParameter)
eventParameter   = eventParameterValue
                 / eventParameterName "=" eventParameter
                 / eventParameterName "(" eventParameters ")"
eventParameterString = 1*(SuitableEventParamCharacter)
```

```
eventParameterName   = eventParameterString
eventParameterValue  = eventParameterString / quotedString
; For NCS (and MGCP)
DigitMap           = DigitString  / "(" DigitStringList ")"
DigitStringList    = DigitString 0*( "|" DigitString )
DigitString        = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition      = DigitMapLetter / DigitMapRange
; NOTE "X" is now included
DigitMapLetter     = DIGIT / "#" / "*" / "A" / "B" / "C" / "D" / "T"
                   / "X" / ExtensionDigitMapLetter
ExtensionDigitMapLetter = "E" / "F" / "G" / "H" / "I" / "J" / "K"
                        / "L" / "M" / "N" / "O" / "P" / "Q" / "R"
                        / "S" / "U" / "V" / "W" / "Y" / "Z"
; NOTE "[x]" is now allowed in MGCP.
; In NCS, only the "x" form is allowed
DigitMapRange = "[" 1*DigitLetter "]"
              / "X" ; Added for NCS only
DigitLetter   = *((DIGIT "-" DIGIT) / DigitMapLetter)
ObservedEvents = SignalRequests
EventStates    = SignalRequests
ConnectionParameters = ConnectionParameter
                        0*( "," 0*(WSP) ConnectionParameter )
ConnectionParameter  = ( "PS" "=" packetsSent )
                     / ( "OS" "=" octetsSent )
                     / ( "PR" "=" packetsReceived )
                     / ( "OR" "=" octetsReceived )
                     / ( "PL" "=" packetsLost )
                     / ( "JI" "=" jitter )
                     / ( "LA" "=" averageLatency )
                     / ( ConnectionParameterExtensionName
                             "=" ConnectionParameterExtensionValue )
                     / RemotePacketsSent
                     / RemoteOctetsSent
                     / RemotePacketsLost
                     / RemoteJitter
; NCS and TGCP define the following four connection parameter extension
; names:
RemotePacketsSent  = "PC/RPS" "=" packetsSent
RemoteOctetsSent   = "PC/ROS" "=" octetsSent
RemotePacketsLost  = "PC/RPL" "=" packetsLost
RemoteJitter       = "PC/JI"  "=" jitter
packetsSent     = 1*9(DIGIT)
octetsSent      = 1*9(DIGIT)
packetsReceived = 1*9(DIGIT)
octetsReceived  = 1*9(DIGIT)
packetsLost     = 1*9(DIGIT)
jitter          = 1*9(DIGIT)
averageLatency  = 1*9(DIGIT)
ConnectionParameterExtensionName = VendorCPExtensionName
                                 /    PackageCPExtensionName
VendorCPExtensionName  = "X" "-" 2*ALPHA
                       / NCSTGCPVendorCPExtensionName
;Text below is for NCS and TGCP only. The difference compared to MGCP
;is simply that MGCP requires 2 alpha characters whereas NCS and TGCP
;allow 2 or 3 alpha characters for VendorCPExtensionName
NCSTGCPVendorCPExtensionName = "X" "-" 2*3ALPHA
PackageCPExtensionName = packageName "/" CPName
CPName = 1*(ALPHA / DIGIT / HYPHEN)
ConnectionParameterExtensionValue = 1*9(DIGIT)
MaxMGCPDatagram = 1*9(DIGIT)
ReasonCode = 3DIGIT
             [1*(WSP) "/" packageName]     ; Only for 8xx
             [WSP 1*(%x20-7E)]
```

```
SpecificEndpointID = endpointName
SecondEndpointID   = endpointName
RequestedInfo = infoCode 0*("," 0*(WSP) infoCode)
infoCode = "B" / "C" / "I" / "N" / "X" / "L" / "M" / "R" / "S"
         / "D" ; For NCS (and MGCP) only
    / "O" / "P" / "E" / "Z" / "Q" / "T" / "RC" / "LC"
    / "A" / "ES" / "RM" / "RD" / "PL" / "MD" / extensionParameter
         / "VS" / "ZM" / "ZN" ; NCS and TGCP define these
                              ; three extensionParameters
;NCS and TGCP allows for process and loop control in either order
QuarantineHandling = loopControl / processControl
                   / (loopControl "," 0*(WSP) processControl )
                   / (processControl "," 0*(WSP) loopControl)
loopControl    = "step" / "loop"
processControl = "process" / "discard"
DetectEvents = SignalRequests
RestartMethod = "graceful" / "forced" / "restart" / "disconnected"
              / "cancel-graceful" / extensionRestartMethod
extensionRestartMethod = PackageExtensionRM
PackageExtensionRM     = packageName "/" 1*32(ALPHA / DIGIT / HYPHEN)
RestartDelay = 1*6(DIGIT)
extensionParameter = VendorExtensionParameter
                   / PackageExtensionParameter
                   / OtherExtensionParameter
VendorExtensionParameter  = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
PackageExtensionParameter = packageName "/"
                            1*32(ALPHA / DIGIT / HYPHEN)
; must not start with "x-" or x+"
OtherExtensionParameter   = 1*32(ALPHA / DIGIT / HYPHEN)

;If first character is a double-quote, then it is a quoted-string
parameterString = (%x21 / %x23-7F) *(%x20-7F) ; first and last must not
                                              ; be white space
                   / quotedString
MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                                      *2(EOL *SDPinformation)
MGCPResponseLine = responseCode 1*(WSP) transaction-id
                        [1*(WSP) "/" packageName]    ; Only for 8xx
                             [WSP responseString] EOL
responseCode = 3DIGIT
responseString = *(%x20-7E)
SuitablePkgExtLCOCharacter =  SuitableLCOCharacter
SuitableExtLCOCharacter = DIGIT / ALPHA / "+" / "-" / "_" / "&"
              / "!" / "'" / "|" / "=" / "#" / "?"
              / "." / "$" / "*" /       "@" / "[" / "]"
              / "^" / "`" / "{" / "}" / "~"
SuitableLCOCharacter   = SuitableExtLCOCharacter / "/"
SuitableExtLCOValChar  = SuitableLCOCharacter / ":"
; VCHAR except """, "(", ")", ",", and "="
SuitableEventParamCharacter = %x21 / %x23-27 / %x2A-2B
                            / %x2D-3C / %x3E-7E
; NOTE: UTF8 encoded
quotedString  = DQUOTE 0*(quoteEscape / quoteChar) DQUOTE
quoteEscape   = DQUOTE DQUOTE
quoteChar = (%x00-21 / %x23-FF)
EOL = CRLF / LF
HYPHEN = "-"
; See RFC 2327 for proper SDP grammar instead.
SDPinformation = SDPLine CRLF *(SDPLine CRLF)        ; see RFC 2327
SDPLine        = 1*(%x01-09 / %x0B / %x0C / %x0E-FF) ; for proper def.
```

Appendix VII

Electronic surveillance

VII.1 MGC

The format for the electronic surveillance parameters in the LocalConnectionOptions of a CRCX or a MDCX command sent to the MG is:

- The Call Content connection Identifier encoded as the keyword "es-cci" followed by a colon and a string of up to 8 hex characters corresponding to a 32-bit identifier for the Call Content Connection Identifier.
- The Call Content Destination encoded as the keyword "es-ccd" followed by a colon and an IP-address encoded similarly to an IP-address for the domain name portion of an endpoint name. The IP-address is followed by a colon and up to 5 decimal characters for a UDP port number to use.

A MGC MUST include both the "es-cci" and the "es-ccd" parameters in the LocalConnectionOptions of a CRCX or MDCX when notifying a MG of electronic surveillance parameters.

The following is an example of a CRCX command with electronic surveillance parameters:

```
CRCX 1204 ds/ds1-1/1@mg.cablelabs.com MGCP 1.0 TGCP 1.0
C: 5678ABCD
L: p:10, a:PCMU, es-cci:123456, es-ccd:[128.96.41.1]:3456
M: sendrecv
X: 1237
```

The following is an example of a MDCX command with electronic surveillance parameters:

```
MDCX 1206 ds/s-1/ds1-1/1@mg.cablelabs.com MGCP 1.0 TGCP 1.0
C: 5678ABCD
I: 32F345E2
L: p:10, a:PCMU, es-cci:123456, es-ccd:[128.96.41.1]:3456
M: sendrecv
X: 1238
```

VII.2 MG

When a MG receives a CRCX with a non-empty "es-cci" and a non-empty "es-ccd" in the LCO, the MG MUST commence replicating, forwarding, and encapsulating all packets that are received and transmitted on the connection. The process of replicating, forwarding, and encapsulating all packets on a connection is known as Call Content Surveillance. A MG that is performing Call Content Surveillance MUST replicate, forward, and encapsulate all packets that it generates for the connection. A MG that is performing Call Content Surveillance MUST replicate, forward, and encapsulate all packets that it receives on a connection. The packets that are encapsulated MUST be identical to the packets on the connection. The MG MUST encapsulate all replicated and forwarded packets with the Call Content Connection Identifier contained in the "es-cci" field of the LCO. The MG MUST forward all replicated and encapsulated packets to the IP address and UDP port indicated in the "es-ccd" field of the LCO.

A MG performing Call Content Surveillance MUST terminate Call Content Surveillance when either:

1) the connection is terminated due to a DLCX from the MGC;
2) the connection is terminated due to a DLCX from the MG;

3) the connection is terminated due to internal error conditions such as:
   - The MG suffers from component failure;
   - The DS1 that the endpoint is on goes out of service;
4) the MG receives a MDCX with an empty "es-cci" field ("es-cci:") or with an empty "es-ccd" ("es-ccd:") field.

If a MG that is performing Call Content Surveillance receives a MDCX with a LCO that contains valid and new "es-cci" or "es-ccd" parameters, the MG MUST use the new "es-cci" or "es-ccd" parameters when performing Call Content Surveillance.

If a MG receives a CRCX or a MDCX with "es-cci" and "es-ccd" fields, but does not support electronic surveillance, but is able to execute the command except for the electronic surveillance portion, the MG MUST return a 210 – the requested transaction was executed normally, but the gateway does not support electronic surveillance – response code.

If a MG receives a CRCX or a MDCX with "es-cci" and "es-ccd" fields, and does support electronic surveillance, and is able to execute the command except for the electronic surveillance portion due to resource constraints, the MG MUST return a 211 – the requested transaction was executed normally, but the gateway could not perform electronic surveillance because it does not have sufficient resources – response code.

If a MG receives a CRCX or a MDCX with a LCO that contains only a "es-cci" or "es-ccd" parameter, but not both, and is able to execute the command except for the electronic surveillance portion, the MG MUST return a 212 – the requested transaction was executed normally, but all necessary electronic surveillance parameters were not in LCO – response code.

If a MG that is not performing Call Content Surveillance on a connection receives a MDCX with "es-cci" or "es-ccd" fields for that connection, the MG MUST return a 213 – the requested transaction was executed normally, but electronic surveillance cannot be started mid-call – return code.

If a MG receives a CRCX or a MDCX with unusable "es-cci" or "es-ccd" fields, and the MG is able to execute the command except for the electronic surveillance portion, the MG MUST return a 214 – the requested transaction was executed normally, but the electronic surveillance parameters were unrecognized – return code. If the MG receiving the MDCX with the unusable "es-cci" or "es-ccd" parameters was already performing Call Content Surveillance, then the MG MUST continue performing Call Content Surveillance and ignore the "es-cci" and "es-ccd" parameters in the MDCX.

When a MG receives an AuditEndpoint command with the capabilities parameter, the MG MUST return the keyword "es" if electronic surveillance is supported.

When a MG that is performing Call Content Surveillance receives an AuditConnection command for that connection with the LocalConnectionsOptions being one of the parameters being audited, the MG MUST return in the LCO the "es-cci" and "es-ccd" parameters that the MG is currently using to perform Call Content Surveillance.

Appendix VIII

Example event packages

VIII.1 MF FGD Operator Services package

Package name: MO

The codes in Table VIII.1 are used to identify events and signals for the "MO" package for one-way outgoing MF Operator Services trunks "Operator Services Signalling". MF FGC Operator Services signalling is supported as well. This package will be used for general operator service trunks as well as dedicated emergency services trunks:

Table VIII.1/J.171.1 – Codes used to identify no package events and signals

Code	Description	Event	Signal	Additional information
ans	Call answer	P	–	
ft	Fax tone	√	–	
ld	Long duration connection	C	–	
mt	Modem tone	√	–	
orbk	Operator ringback	√	–	
rbz	Reverse make busy	P	–	
rcl	Operator recall	–	BR	
rel	Release call	P	BR	
res	Resume call	–	BR	
rlc	Release complete	P, S	BR	
sup(<addr>, <id>)	Call set-up	–	TO	Variable time-out
sus	Suspend call	–	BR	
swk	Start wink	√	–	
TDD	Telecommuniations Device for the Deaf (TDD) tones	√		
oc	Operation complete	√		
of	Operation failure	√		

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

Call answer (ans): Call Answer occurs at the time of the OSS ANI request, i.e., the call may not necessarily have been cut-thru to an operator. After Call Answer occurs, facility-hold will be established, i.e., only the OSS can now release the trunk.

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

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.

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

Operator ringback (orbk): This event will be generated when the OSS requests that the calling party be alerted[32].

Reverse make busy (rbz): This event occurs when the OSS marks the trunk. A release event will be generated when the trunk is no longer busy.

Operator recall (rcl): This signal may be applied to invoke operator recall, e.g., due to customer hook-flash to bring the operator back.

Release call (rel): Release call may be signalled to the media gateway; however, if facility-hold is established, then the call will not be disconnected until the OSS releases it. The media gateway generates a "release call" event when the OSS is considered to have released the trunk. In this case, the event may be parameterized with one of the cause codes in Table VIII.2 indicating the reason for the release:

Table VIII.2/J.171.1 – Release call cause codes

Cause code	Reason
0	Normal release
3	No route to destination
8	Preemption
19	No answer
21	Call rejected
27	Destination out of order
28	Invalid number format (e.g., address incomplete)
38	Network out of order
111	Protocol/signalling error, unspecified (e.g., time-out)

Resume call (res): This signal indicates that the other party resumed the call, i.e., the party went off-hook.

Release complete (rlc): The endpoint and MGC use the release complete event/signal to confirm the call has been released and the trunk is available for another call.

Call setup (sup(<addr>, <id>)): Set up a call to the operator service system using the address and identification information provided. The address information will be of the form:

addr(MF_1, MF_2, …, MF_n)

and the identification information will be of the form:

id(MF_1, MF_2, …, MF_n)

where each of MF_i will be one of the following MF digit symbols in Table VIII.3:

[32] If the calling party is on-hook, ringing will typically be applied, where as reorder tone will typically be applied in case the calling party is off-hook.

Table VIII.3/J.171.1 – MF digit symbols

Symbol	MF digit	Symbol	MF digit
0	MF 0	K0	MF K0 or KP
1	MF 1	K1	MF K1
2	MF 2	K2	MF K2
3	MF 3	S0	MF S0 or ST
4	MF 4	S1	MF S1
5	MF 5	S2	MF S2
6	MF 6	S3	MF S3
7	MF 7	K0	MF K0 or KP
8	MF 8		
9	MF 9		

Thus, an example call set-up signal could be:

```
sup(addr(K0, 5,5,5,1,2,1,2, SO), id(K0, 5,5,5,1,2,3,4, SO))
```

Suspend call (sus): This signal indicates that the other party suspended the call, i.e., the party went on-hook.

Start wink (swk): A media Gateway Controller can request the Media Gateway to notify it when the wink start signal occurs.

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

Operation complete (oc): The operation complete event is generated when the gateway was asked to apply one or several signals of type TO on the endpoint, and one or more of those signals completed without being stopped by the detection of a requested event such as off-hook transition or dialed digit. The completion report may carry as a parameter the name of the signal that came to the end of its live time, as in:

```
O: MO/oc(MO/sup)
```

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.

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: MO/of(MO/sup)
```

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

VIII.2 MF Terminating Protocol package

Package name: MT

In this version of the TGCP Recommendation, the package can only be used for busy-line verification (BLV) and operator interrupt (OI) on one-way incoming MF terminating trunks dedicated to BLV and OI[33].

The codes in Table VIII.4 are used to identify events and signals for the "MT" package for one-way incoming "MF Terminating trunk circuits" used for BLV and OI:

Table VIII.4/J.171.1 – Codes used to identity MT package events and signals

Code	Description	Event	Signal	Additional information
ans	Call answer	–	BR	
bz	Busy tone	–	TO	Time-out = 30 seconds
hf	Hook-flash	–	BR	
inf	Information digits	√		
oc	Operation complete	√	–	
of	Operation failure	√	–	
oi	Operator interrupt	√	–	
pst	Permanent signal tone	–	TO	Time-out = infinite
rel	Release call	P	BR	
res	Resume call	–	BR	
rlc	Release complete	P, S	BR	
ro	Reorder tone	–	TO	Time-out = 30 seconds
sup	Call set-up	P	–	
sus	Suspend call	–	BR	

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

NOTE – For specific technical details of the tones used, see ITU-T Rec. E.180/Q.35.

Call answer (ans): The call answer signal informs the endpoint that the verified party has answered. This includes the case where the verified party was already off-hook. The endpoint is expected to pass on answer supervision to the OSS.

Busy tone (bz): Station Busy.

Hook flash (hf): This signal indicates that the verified party performed a hook-flash.

Information digits (inf (<inf-digits>): Used on an incoming MF trunk to indicate digits received. The parameter value <inf-digits> are all of the digits accumulated up to and including the digit delimiter, i.e., ST, ST', ST", or ST'".

The value of <inf-digits> is a comma-separated list of MF digits:

$MF_1, MF_2, \ldots, MF_n$

[33] Note that when Operator Services are provided by an off-net provider, the OSS may not have access to subscriber databases to determine whether BLV and OI should be allowed or not.

where each of the MF_I will be one of the following MF digit symbols in Table VIII.5:

Table VIII.5/J.171.1 – MF digit symbols

Symbol	MF digit	Symbol	MF digit
0	MF 0	K0	MF K0 or KP
1	MF 1	K1	MF K1
2	MF 2	K2	MF K2
3	MF 3	S0	MF S0 or ST
4	MF 4	S1	MF S1
5	MF 5	S2	MF S2
6	MF 6	S3	MF S3
7	MF 7	K0	MF K0 or KP
8	MF 8		
9	MF 9		

Thus, an example signal or event might look like:

```
inf(k0, 5,5,5,1,2,3,4, s0)
```

An example where the inter-digit timer expired after the 5,5,5 would appear as follows:

```
inf(k0, 5,5,5)
```

Operation complete (oc): See the definition of "operation complete" in the ISUP trunk package.

Operation failure (of): See the definition of "operation failure" in the ISUP trunk package.

Operator interrupt (oi): The operator interrupt event occurs when the operator attempts to interrupt the call and generates the "operator interrupt" tone. Since no standard tone is defined for this, the event is here defined to occur when a certain level of energy is detected on the trunk corresponding to a transition from line noise to voice or tones. It should be noted that it is hereby not possible to detect a transition back to line noise from voice/tones.

Permanent signal tone (pst): **Release call (rel)**: The MGC may use the release signal to release the call[34]. In this case, the release signal may not be parameterized.

The endpoint may in turn use the event to inform the MGC that it has released the call – in this case the event may be parameterized with one of the cause codes in Table VIII.6 indicating the reason for the release:

[34] Note that the verifying operator normally controls release of completed no-test connections and the suspend signal should thus typically be used.

Table VIII.6/J.171.1 – Release call cause codes

Cause code	Reason
0	Normal release
3	No route to destination
8	Preemption
19	No answer
21	Call rejected
27	Destination out of order
28	Invalid number format (e.g., address incomplete)
38	Network out of order
111	Protocol/signalling error, unspecified (e.g., time-out)

Resume call (res): This signal indicates that the verified party resumed the call, i.e., the party went off-hook.

Release complete (rlc): The endpoint and MGC use the release complete event/signal to confirm the call has been released and the trunk is available for another call.

Call set-up (sup): A "sup" event is used to indicate when an incoming call arrives (corresponding to the incoming off-hook event). The event is provided without parameters.

Suspend call (sus): This signal indicates that the verified party suspended the call, i.e., the party went on-hook.

BIBLIOGRAPHY

– *Bellcore Notes on the Networks*, Bellcore, SR-2275.

– *Compatibility Information for Feature Group D Switched Access Service,* Bellcore, TR-NPL-000258, Issue 1, October 1985.

– *Interoffice LATA Switching Systems Generic Requirements (LSSGR): Verification Connections (25-05-0903)*, Bellcore, TR-TSY-000531, Issue 2, July 1987.

– *Signalling for Analog Interfaces*, Bellcore, LSSGR GR-506-CORE, Issue 1, June 1996.

– *Switching System Generic Requirements for Call Control Using the Integrated Services Digital Network User Part (ISDNUP)*, Bellcore, LSSGR GR-317-CORE, Issue 2, December 1997.

– *Custom Call-Handling Features (FSD 80 Series)*, Bellcore, OSSGR GR-1176-CORE, Issue 1, March 1999.

– IETF RFC 1827 (1995), *IP Encapsulating Security Payload (ESP)*.

– IETF RFC 2974 (Experimental, 2000), *Session Announcement Protocol*.

– *RTP Parameters*, http://www.iana.org/assignments/rtp-parameters.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
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	Telecommunication management, including TMN and network maintenance
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, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2006