Implementation of Call Agent
Using the
Network-based Call Signaling Protocol

By
Saira Viqar
Samana Zehra
Nadia Rahim

Dissertation for partial fulfillment of the requirements of MCS/NUST for the award of the B.E degree in Software Engineering

Department of Computer Science
Military College of Signals
National University of Sciences and Technology
Rawalpindi

October 2001

Abstract

This undergraduate dissertation describes the implementation of a basic constituent of Voice-over-IP (VoIP) system, the Call Agent, using the PacketCable “Network-based Call Signaling” (NCS) protocol. It discusses NCS as a call signaling protocol for use in centralized call control architecture. The capability of the NCS Call Agent of call control, connection control, auditing, and management of resources of its media gateways and its implementation is also discussed. All the basic features of the protocol except for the security and Quality of Service (QoS) have been described and implemented.

Acknowledgements

We would like to thank Dr. Farrukh Kamran at Enabling Technologies for putting us on the right track and guiding us in understanding the concepts of Voice-over-IP (VoIP) and the Network-based call Signaling Protocol (NCS). We would like to thank all our teachers especially Mr. Ali Ahsan for guiding us in solving our problems related to Java language and our project supervisor Lt. Col. Mohammed Tufail for his cooperation and support to bring this project to completion.

We would also like to thank our families and friends for their continuous encouragement and moral support.

Declaration

"No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere"

Preface

HOW TO USE THIS DOCUMENT

Refer to	For…..
Chapter 1	Overview of VoIP and Call Agent
Chapter 2	Project Specifications
Chapter 3	Details of the Network-based Call Signaling (NCS) Protocol
Chapter 4	UML Class design of the Software
Chapter 5	Implementation details
Summary	Achieved results and future Recommendations
Appendix A	Events specified in the Event Packages of NCS Protocol
Appendix B	Return Codes and Error Codes
Appendix C	Reason Codes
Appendix D	Example Call Flow
Appendix E	Formal syntax description of the NCS Protocol

Table of Contents

Chapter 1

Introduction

1 BACKGROUND

1.1 Overview of VoIP

Voice-over-IP (VoIP) implementations enable users to carry voice traffic, for example, telephone calls, faxes and related data) over an IP network.

There are 3 main causes for the evolution of the Voice over IP market:

- Low cost phone calls
- Add-on services and unified messaging
- Merging of data/voice infrastructures

A VoIP system consists of a number of different components like Gateway/Media Gateway, Gatekeeper, Call agent, Media Gateway Controller, Signaling Gateway and a Call manager.

In VoIP, the voice signals are segmented into frames and stored in voice packets. These voice packets are encoded at the sending gateway according to the standard coding-decoding schemes, transported using IP in compliance with one of the protocol specifications for transmitting multimedia (voice, video, fax and data) across a network and decoded at the receiving gateway side. The conversation proceeds using Real-Time Transport Protocol/User Datagram Protocol/Internet Protocol (RTP/UDP/IP) as the protocol stack.

1.2 VoIP Protocols

- H.323
- IPDC: Internet Protocol Device Control
- MGCP/NCS: Media Gateway Control Protocol/Network-based Call Signaling
- RVP over IP
- SAPv2: Session Announcement Protocol
- SDP: Session Description Protocol
- SGCP: Simple Gateway Control Protocol
- SIP: Session Initiation Protocol
- Megaco (H.248)

1.3 Services of VoIP

The following are some examples of services provided by a Voice over IP network according to market requirements:

- Phone to phone
- PC to phone
- phone to PC
- fax to e-mail
- e-mail to fax
- fax to fax
- voice to e-mail
- IP Phone
- call center applications
- VPN
- Unified Messaging
- Wireless Connectivity
- IP PABX
- soft switch implementations

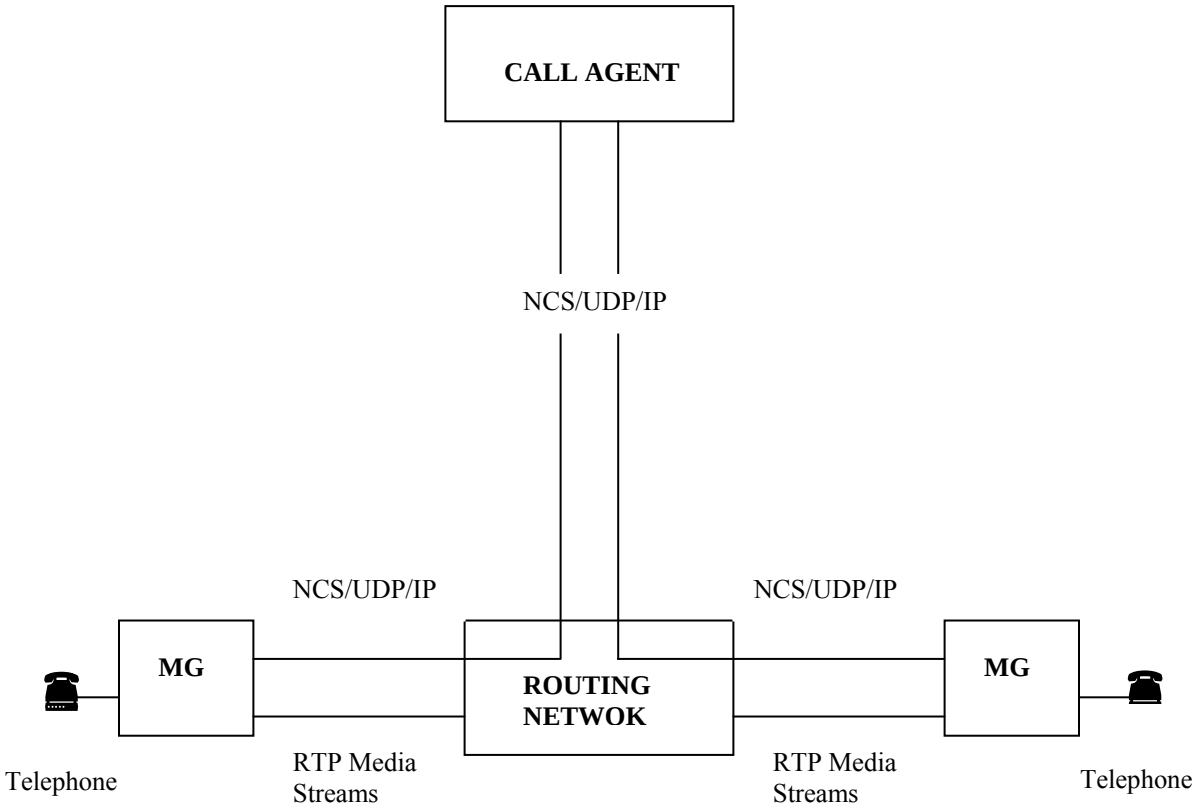
2 WHAT IS A CALL AGENT AND WHAT DOES IT DO?

The Call Agent is one of the basic components of the VoIP System. It is defined as a server which controls and serves media gateways that support users and their subscribed telephony features.

The Call Agent implementation discussed in this dissertation ihas been done using one of the VoIP protocols, NCS or the Network-based Call Signaling Protocol. NCS is based on the Media Gateway Control Protocol (MGCP) [2] which was the result of a merge of the Simple Gateway Control Protocol [7], and the IP Device Control (IPDC) family of protocols [8]. It is a profile of MGCP with a few minor extensions and modifications. NCS is used to control VoIP media gateways or telephony gateways. A telephony gateway is a network element that provides conversion between the audio signals carried on telephone circuits and data packets carried over the Internet or over other packet networks.

NCS has a *call control architecture* -it separates the control functions of a phone call from the actual media flow: call control performed by Call Agent whereas media flow handled by media gateway.

The various entities involved in the NCS architecture are shown in the following diagram (reference [1]):



The users device – the telephone is connected to the gateway through an analog line. The gateway is acting as a translator or interface between the analog lines on one side and a packet switched network on the other.

The Call Agent is controlling the media gateways abbreviated as MG and is responsible for creating and managing connections between the endpoints(analog access lines). There are two different paths, one for the control signals which are being exchanged between the gateway and the call agent, the protocol stack being NCS over UDP over IP. The other path is the one along which the data or voice packets are traveling. The protocol stack here is RTP over UDP over IP.

Call Agent has the *"intelligence"* of call control operations:

Call Agents instruct the gateways to create connections between endpoints and to detect certain events, e.g., off-hook, and generate certain signals, e.g., ringing. It is strictly up to the Call Agent to specify how and when connections are made,

between which endpoints as well as what events and signals are to be detected and generated on the endpoints.
NCS is a *master/slave protocol*, where the gateways are expected to execute commands sent by the Call Agents [6]. All the intelligence and decision making capability resides within the Call Agent and it must provide the appropriate commands to control the gateways. Gateway control is achieved through a *set of Transactions* (a transaction is a command & its mandatory response).

The NCS assumes a *connection model* where the basic constructs are *endpoints* and *connections* (Reference [3] for detail). A gateway contains a collection of endpoints, which are sources, or sinks of data. An example of a physical endpoint is an interface on a gateway that terminates an analog POTS connection to a phone. A *Connection* is point-to-point or multipoint, and is an association between two endpoints or more 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.

NCS has a centralized architecture. One central entity is controlling all the other entities which means implementation is simpler.

Chapter 2

Project Specifications

2.1 Statement

The Call Agent will be based on the Network-based Call Signaling (NCS) Protocol, which is an application-layer control (signaling) protocol for creating, modifying and terminating calls and connections with one or more user-ends/endpoints.

The proposed system will be able to control residential gateways/embedded clients (a gateway that terminates analog access lines to phones). It will be able to exchange different commands and responses with the gateways to achieve the following:

- Implementation of basic call flow (between two parties).
    - Call setup: creating the connection and modifying it when required.
    - Call termination: deleting connections.
    - Translation of collected digits (dialed by user) into a destination IP address (through an Address-Resolver).
- Detection of events on the endpoints/user end, like off-hook, on-hook, flash-hook and get their report
- Asking the gateway to apply different signals like dial-tone, ringing, ringback-tones, call-waiting-tones on the user-end.
- Determination of endpoint capabilities.
- Keeping track of endpoint state.
- Handling error conditions, e.g., User has dialed a number corresponding to an endpoint, which the Call Agent does not know.

The proposed system will support two Gateways. Each Gateway will support 4 endpoints but the implementation will be capable of handling n number of endpoints. The gateways will be simulated to fulfil the testing requirements of the Call Agent

and thus will not be a complete implementation. The user end/endpoint will be simulated in the form of a GUI of telephone on the gateways.

2.2 Design Tool/Language

Unified Modeling Language (UML) has been used to software design and Rational Rose 98 has been used as the design tool.

2.3 Development Language

The software is written in Java.

2.4 Development Environment

The system has been developed in Kawa 4.0 and compiled and executed with Java Development Toolkit version 1.3 (jdk 1.3).

2.5 Platforms Supported

Microsoft Windows 95/98, Microsoft Windows NT 4.0, Microsoft Windows 2000, and Microsoft Windows me.

Chapter 3

Introduction to NCS Protocol

3.1 MODEL AND NAMING CONVENTIONS

NCS assumes a connection model where the basic constructs are endpoints and connections. Connections are grouped into calls. One or more connections can belong to a call. Connections are set up at the initiative of Call agent. For more details refer to [3].

3.1.1 Endpoint Names

The endpoint name consists of two parts, both of which are case insensitive:

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

General form:

```
local-endpoint-name@domain-name
```

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

```
MyEmbeddedClient.cablenetwork.com
```

Each of the endpoints associated with the embedded clients is identified by a separate local endpoint name.

General rules for Local Endpoint Name:

The local endpoint name is hierarchical where the least specific component of the name is the leftmost term, and the most specific component is the rightmost term. The local endpoint name must adhere to the following rules:

- The individual terms of the local endpoint name must be separated by a single slash (“/”). The individual terms are ASCII character strings composed of letters, digits or other printable characters, with the exception of characters used as delimiters in endpoint-names (“/”, “@”), characters used for wildcarding (“*”, “$”), and white space characters. Wild carding is represented either by an asterisk

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

```
term1/term2/term3
```

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

```
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 embedded client in question".

- A term represented by a dollar sign is to be interpreted as:
  "*use any one* value of this term known within the scope of the embedded client in question".

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

3.1.1.1 Embedded Client Endpoints Names

`Types of Endpoints:`
Embedded clients support the following two endpoint-types:

1. Analog telephone:

   The Analog Telephone is represented as an analog access line (aaln). This is basically the equivalent of an analog telephone line as known in the PSTN

2. Video:

   The details of the video device-type have not been provided in the current version of the NCS.

3.1.1.1.1 *Analog Access Line Endpoints:*

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

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

Example analog access line local endpoint names:

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

The number of endpoints and their type is determined from the provisioning/(auto)configuration process. The endpoint-type is derived from the local portion of the endpoint name, such as analog phone or videophone.

3.1.1.1.2 *Video Endpoints*

The details on video endpoints is not provided in the current version of NCS.

3.1.2 Call Names

Calls are identified by platform independent unique identifiers. Call identifiers are hexadecimal strings with a maximum length of 32 characters and are created by the Call Agent.

When a Call Agent builds several connections that pertain to the same call, either on the same gateway or in different gateways, all these connections will be linked to the same call through the call identifier. This identifier then can be used by accounting or management procedures.

3.1.3 Connection Names

Connection identifiers are created by the gateway when it is requested to create a connection. They identify the connection within the context of an endpoint. A connection identifier is a hexadecimal string with the maximum length of 32 characters. 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.

3.1.4 Call Agent Name

Call Agent name consists of two parts just like the endpoint names:

- Local portion of the name
- Domain name

The local name part does not exhibit any internal structure.

General form:

```
local-name@domain-name
```

Example:

```
cal@ca.whatever.net
```

3.1.5 Events/Signals Names

A Call Agent may ask to be notified about certain events occurring at an endpoint, e.g., off-hook event. It may also request certain signals to be applied to an endpoint,

 e.g., dial-tone.

Events and Signals are grouped into packages where a package is a collection of endpoints supported a particular endpoint-type.The endpoints of type analog access lines (aaln) support *Line package (L) and Analog Display Services Interface-ADSI package (S).* The default package is L.

Event names consist of a package name and an event code. Both are case insensitive strings of letters, digits and hyphens with the restriction that hyphen is never the first or last character. The package name is separated from the event

name by a slash("/"). The package name can be excluded in case default package is used.For example,

```
L/dl      dial-tone in line package for analog access line.
dl        dial-tone in line package (default) for analog
          access line.
```

The line package definition of the line package is given in Appendix A.

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

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

But, the package name "*" MUST NOT be assigned to a package, and the event code "all" MUST NOT be used in any package.

Most of the events and signals are detected and generated on endpoints but some events and signals may be detected and generated on the connections, e.g., an endpoint may be asked to provide a ringback tone on a connection. For example,

```
L/rt@0A895      ringback tones in line package at connectionId
                0A895
```

The "*" and "$" wildcards can be used to denote "all connections" and the "current connection". This convention MUST NOT be used unless the event notification request is "encapsulated" within a CreateConnection or ModifyConnection command.

```
L/ma@*          media start in line package at all connections
L/rt@$          ringback tones in line package at current
                connection being created or modified
```

3.2 NOTIFIED ENTITY

The NCS protocol gives the concept of "notified entity". Notified entity for an endpoint is the Call Agent that is controlling it. When the endpoint needs to send a command to the Call Agent, it must send the command to the "notified entity". Most of the commands sent by the Call Agent to the gateways explicitly name the notified entity through the use of a "Notified Entity" parameter.

3.3 DIGIT MAPS

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

An alternative procedure involves the gateway notifying the Call Agent of the dialed digits as soon as they are dialed, a.k.a., overlap sending. However, such a procedure generates a large number of interactions. It is preferable to accumulate the dialed numbers in a buffer, and then to transmit them in a single message. The problem with this accumulation approach, however, is that it is difficult for the gateway to predict how many numbers it needs to accumulate before transmission.

The solution to this problem is to load the gateway with a digit map that corresponds to the dial plan. For example, a dial plan in given the following digit map:

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

Each string in the above dial plan is an alternate numbering scheme. The formal syntax of the digit map is described by the ABNF notation in Appendix E where the formal syntax description of the protocol is given.

A gateway that detects digits, letters, or timers will:

1. Add the event parameter code for the digit, letter, or timer, as a token to the end of the “current dial string” internal state variable.

2. Apply the “current dial string” to the digit map table, attempting a match, in lexical order, to each regular expression in the Digit Map.

3. If the result is under-qualified (partially matches at least one entry in the digit map), do nothing further.

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

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

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

timer T can be used as an inter-digit timer when overlap sending is used. When used with a digit map, timer T takes on one of two values, Tpar or Tcrit. When at least one more digit is required for the digit string to match any of the patterns in the digit map, timer T takes on the value Tpar, corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value Tcrit corresponding to critical timing. When timer T is used without a digit map, timer T takes on the value Tcrit. The default value for Tpar is 16 seconds and the default value for Tcrit is 4 seconds. The provisioning process may alter both of these.

Digit maps can be provided to the gateway by the Call Agent, whenever the Call Agent instructs the gateway to listen for digits.Digit Maps, when provided by the Call Agent, MUST be as defined in this section.

3.4 DESCRIPTION OF NCS COMMANDS

This section (see [3] for more details) describes the commands of the NCS in the form of a remote procedure call (RPC) like API, which can be referred to as the gateway control functions. A gateway control functions is defined for each NCS command, where the function takes and returns the same parameters as the corresponding NCS command. The following is an overview of the commands:

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

- The gateway will then use the Notify command to inform the Call Agent when the requested events occur on the specified endpoint/

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

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

- The Call Agent can use the DeleteConnection command to delete an existing connection. In some circumstances, the DeleteConnection command also can be used by a gateway to indicate that a connection can no longer be sustained.

- The Call Agent can use the AuditEndpoint and AuditConnection commands to audit the status of an “endpoint” and any connections associated with it. Network management beyond the capabilities provided by these commands are generally desirable, e.g., information about the status of the embedded client.

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

3.4.1 NotificationRequest

The NotificationRequest command is used to request the gateway to send a notification upon the occurrence of certain events at an endpoint.

For example, the CallAgent may request the gateway to send a notification when an off-hook event occurs at a certain specified endpoint. When the CallAgent receives this notification it may decide to create a connection at that endpoint.

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

`EndpointId`

This is the Identifier for the endpoint(s) for which the event is to be detected.
The “any of “ wildcard cannot be used, but the “all of” wild card must be supported.

`NotifiedEntity`

This specifies a new “notified entity” for an endpoint. The “notified entity” for an endpoint is the entity to which the notification is sent once an event occurs at the endpoint. There can be only one “notified entity” for an endpoint at one time.

`RequestIdentifier`

This is used to correlate the request (NotificationRequest command) with the notification it triggers. The Notify command corresponding to a NotificationRequest contains the same RequestIdentifier as that NotificationRequest.

`SignalRequests`

This contains the signals which should be applied to the endpoint(s) by the gateway. Some signals can be applied to connections.
Examples are:

- Dial tone
- Ringing

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

There are three different types of signals:

- ON/OFF (OO)

Once applied, these signals remain until they are explicitly turned off. This can only be if a new SignalRequest is sent in which the signal is turned off. An example is a visual message waiting indicator (VMWI).

- TIME-OUT (TO)

These signals last until they are cancelled (by the occurrence of an event, or by not being included in a subsequent SignalRequests) or a specific period of time has elapsed. A signal that times out will generate an “operation complete” event. Time-out signals have a default time-out value defined for them which can be changed by the provisioning process. The time-out period can also be given as a parameter with the signal . A time-out signal which fails before generating an “operation complete” event, will generate an “operation failure” event.

- BRIEF (BR)

These signals have a short duration and they stop on their own. If a new SignalRequests is received or a signal-stopping event occurs, BR signals do not stop, however, pending BR signals are cancelled. An example of a BR signal is a DTMF digit.

One signal must not occur more than once in a SignalRequests.
The SignalRequests parameter supplied can also be empty. The signals in a new SignalRequests completely replace the currently active time-out signals. Signals not provided in the new list must stop, and currently active time-out signals included in the new list must continue without any effect on their timer.

RequestedEvents

This is a list of events that the gateway is requested to detect at an endpoint. Some events can be detected on a connection. The following are examples of events:

- On-hook transition (when the user hangs up the handset)
- Off-hook transition (when the user picks up the handset)
- Flash-hook (when the user briefly presses the phone hook)
- DTMF digits

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

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

- Accumulate according to Digit Map ,
- Ignore the event ,
- Keep Signal(s) active ,
- Embedded NotificationRequest ,
- Embedded ModifyConnection .

There are two types of events

- PERSISTENT

These events are always detected on an endpoint, whether they are included in the list of RequestedEvents or not. The RequestIdentifier for the Notify will be that of the current NotificationRequest. They can be viewed as always implicitly being included in the list of RequestedEvents with an action to Notify. Examples are off-hook and on-hook events

- NON-PERSISTENT

These are events which have to be explicitly included in the list of RequestedEvents.

The RequestedEvents can be completely empty. The new list of RequestedEvents completely replaces the previous. An event cannot appear more than once in a list of RequestedEvents.
More than one action can be specified for an event, but a given action cannot be specified more than once for a given event. The following table gives the legal combination of actions:

	Notif-y	Accumulate	Accumulate according to digit map	Ignore	Keep Signal(s) Active	Embedded Notification Request	Embedded Modify Connection
Notify	-	-	-	-	✔	-	✔
Accumulate	-	-	-	-	✔	✔	✔
Accumulate According to digit map	-	-	-	-	✔	-	✔
Ignore	-	-	-	-	✔	-	✔
Keep Signal(s) Active	✔	✔	✔	✔	-	✔	✔
Embedded Notification Request	-	✔	-	-	✔	-	✔
Embedded Modify Connection	✔	✔	✔	✔	✔	✔	-

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

combination of actions).

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

`DigitMap`

This parameter works with the “Accumulate according to digit map” action. This is a string according to which digits are collected at the gateway when the “Accumulate according to digit map” action is provided in the RequestedEvents. The DigitMap is persistent and need not be provided each time that the “Accumulate according to digit map action is specified. However, it must be provided in the first NotificationRequest sent to an endpoint.

Each endpoint has a variable called the ***“current dial string”*** in which the digits to be accumulated are collected. Whenever a new NotificationRequest is to be processed, the “current dial string” is initialized to a null string.

The signals being applied by SignalRequests are synchronized with the detection of events specified (implicitly or explicitly) by the RequestedEvents. All time-out signals stop as soon as one of the RequestedEvents is detected, unless the “Keep signal(s) active” action is associated with the event. For example if ringing is applied to an endpoint and the RequestedEvents include off-hook, then the ringing will be stopped as soon as the off-hook event is detected. Even if off-hook is not included in the RequestedEvents, the ringing will stop since off-hook is a persistent event. For the “accumulate according to digit map” action, all time-out signals would be stopped when the first digit is detected.

“Keep Signal(s) Active” action
If the “Keep Signal(s) Active” action is associated with an event, then all the currently active time-out signals will continue to be active, without interruption, when the event is detected.

“Embedded NotificationRequest”
If it is desired that signal(s) should be started when an event occurs, then the “Embedded NotificationRequest” action can be used, which may contain new SignalRequests, RequestedEvents and DigitMap. However, it cannot contain another “Embedded NotificationRequest”.When the “Embedded NotificationRequest” is activated, the “current dial string” will be cleared; the list of observed events and the quarantine buffer will be unaffected. The embedded NotificationRequest action allows the Call Agent to set up a “mini-script” to be processed by the gateway immediately following the detection of the associated event. Any SignalRequests specified in the embedded NotificationRequest will start immediately. Considerable care must be taken to prevent discrepancies between the Call Agent and the gateway. However, long-term discrepancies should not occur as new SignalRequests completely replaces the old list of active time-out signals, and BR-type signals always stop on their own. Limiting the number of On/Off-type signals is

encouraged. It is considered good practice for a Call Agent to occasionally turn on all On/Off signals that should be on, and turn off all On/Off signals that should be off.

"Embedded ModifyConnection"
This can be used if connection modes are desired to be changed when an event occurs. It consists of a series of connection mode changes along with the affected connection-id. The wildcard "$" can be used to denote "the current connection", however this notation must not be used outside a connection handling command – the wildcard refers to the connection in question for the connection handling command. 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. 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 - the rest of the connection mode changes must not be attempted, and the previous successful connection mode changes in the list must not be changed either.

"Ignore" action
This can be used to ignore an event e.g. prevent a persistent event from being notified. However the active signals will still be affected by the event.

The specific events and signals that a given endpoint can detect or perform are determined by the list of event packages that are supported by that endpoint. Each package specifies a list of events and signals that can be detected or applied. A gateway that is requested to detect or to apply an event that is not supported by the specified endpoint must return an error (error code 512 or 513 – not equipped to detect event or generate signal).When the event name is not qualified by a package name, the default package name for the endpoint is assumed. If the event name is not registered in this default package, the gateway MUST return an error (error code 522 – no such event or signal).

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

`QuarantineHandling`

There is an event input holding area known as the quarantine buffer associated with each of the endpoints. QuarantineHandling specifies whether the events in this buffer should be processed or discarded. By default, the events are processed.

`DetectEvents`

This 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.

ReturnCode

This is a parameter returned by the gateway and it contains the outcome of the command. It consists of an integer which is optionally followed by commentary.

3.4.2 Notification

When an observed event is to be notified to the Call Agent, the gateway sends a Notification command to it.

ReturnCode
← Notify(EndpointId
[, NotifiedEntity]
, RequestIdentifier
, ObservedEvents)

EndpointId

This is the name of the endpoint where the event has been detected. It must be a fully qualified name, including the domain name of the gateway. Wildcarding must not be used in the local part of the name.

NotifiedEntity

This parameter is optional and it should be the same as the NotifiedEntity parameter of the NotificationRequest which triggered this Notification command. However, the Notification command will be sent to the current NotifiedEntity for this endpoint, regardless of this parameter. If the triggering NotificationRequest did not contain the NotifiedEntity parameter then it will be absent in the Notification.

RequestIdentifier

This parameter is used to correlate the Notification with its triggering NotificationRequest. It is the same as the RequestIdentifier in the NotificationRequest. If no NotificationRequest has been sent then it will be zero. Persistent events will be treated as though they were included in the last NotificationRequest received.

ObservedEvents

This is a list of events that the gateway detected and accumulated because of the “Notify,” or “Accumulate,” or “Accumulate according to digit map” actions. The events are listed in the order in which they were detected. The list con include only persistent events and events included in the triggering NotificationRequest, RequestedEvents parameter. Events detected on a connection will include the name of the connection. The list will contain the events that were either accumulated (but not notified) or accumulated according to digit map (but no match yet), and the final event that triggered the notification or provided a final match in the digit map. It

should be noted that digits are added to the list of observed events as they are accumulated, irrespective of whether they are accumulated according to the digit map or not. For example, if a user enters the digits “1234” and some event E is accumulated between the digits “3” and “4” being entered, the list of observed events would be “1, 2, 3, E, 4”.

`ReturnCode`

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

3.4.3 CreateConnection

This command is used to create a connection.It is used when setting up a connection between two endpoints.

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

`CallId`

This parameter identifies the call or session to which this connection belongs. It can be used to identify calls for reporting and accounting purposes.

`EndpointId`

This is the identifier for the andpoint at which the connection is to be created. It can be fully specified or it can be under-specified by using the “any of ” wildcard. In this case, the gateway will choose the endpoint and return its identifier in the SpecificEndpointId parameter in the response. The “all of” wildcard must not be used.

NotifiedEntity

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

LocalConnectionOptions

This 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*

This is a list of names of compression algorithms (encoding/decoding methods) used to send and receive media on the connection and must be specified with atleast one value. The list of names should be ordered by preference. The endpoint has to choose only one codec according to the preference given, and should also indicate which remaining codecs it is willing to support as alternatives.

- *Packetization Period*

This is specified in milliseconds according to RFC 2327 (the SDP standard [4]) and only one value must be specified. It only pertains to media sent on a connection.

- *Echo Cancellation*

This is an optional parameter and determines whether echo cancellation should be used on the line side or not. It can have the value “on” or “off”.

- *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 A0H applies corresponding to an IP precedence bits setting of five.

- *Silence Supression*

Whether silence suppression should be used or not in the send direction. The parameter can have the value “on” (when silence is to be suppressed) or “off” (when silence is not to be suppressed). The parameter is optional. When the parameter is omitted, the default is not to use silence suppression.

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

RemoteConnectionDescriptor

This is the connection descriptor for the remote side of the connection. It is similar to the LocalConnectionDescriptor. It describes a session according to SDP. It may be null when there is no information about the remote side of a connection. This may

happen because a connection is formed in two halves, one at each of the endpoints involved. When a CreateConnection is sent to the first endpoint, there is no information about the second endpoint. This information is usually provided later through a ModifyConnection command.

The NCS profile currently assumes that the same media parameters apply to a connection in both the send and receive direction. Part of the information in the RemoteConnectionDescriptor is therefore redundant and a potential for inconsistency with the LocalConnectionOptions exists. It is however purely the responsibility of the Call Agent to ensure that it issues coherent commands to each endpoint to ensure that consistent media parameters are specified. If inconsistency is detected by a gateway though, the LocalConnectionOptions will simply take precedence. When codecs are changed during a call, small periods of time may exist where the endpoints use different codes. Embedded clients MAY discard any media received that is encoded with a different codec than what is specified in the LocalConnectionOptions for a connection.

Mode

This indicates the mode of operation for this side of the connection. The options for this parameter are as follows:

- “send only,”
- “receive only”,
- “send/receive”,
- “conference”,
- “inactive”,
- “replicate”,
- “network loopback”,
- “network continuity test”

Some endpoints may not be capable of supporting all modes. If the command specifies a mode that the endpoint does not support, an error MUST be returned (error code 517 – unsupported mode). Also, if a connection has not yet received a RemoteConnectionDescriptor, an error MUST be returned if the connection is attempted to be placed in any of the modes “send only”, “send/receive”, “replicate”, or “conference” (error code 527 – missing RemoteConnectionDescriptor).

ConnectionId

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

LocalConnectionDescriptor

This is a parameter similar to the RemoteConnectionDescriptor. It is returned by the gateway and it contains information about the local side of a connection. It is

basically a session description and contains addresses and RTP ports for “IN” connections according to the SDP standard.

When the gateway receives a CreateConnection command that does not contain a RemoteConnectionDescriptor, it is in an ambiguous situation. Since it has sent a LocalConnectionDescriptor, it can receive packets on the connection. But since it has not yet received a RemoteConnectionDescriptor describing the other side of the connection, it does not know if those packets are valid or not. So it can discard those packets and face the possibility of missing some important announcement, or it can pass them to the endpoint and face the risk that those packets contain garbage. The gateway must behave according to the value of the mode parameter as follows:

If the mode was set to “receive only”, the gateway MUST accept the voice signals received on the connection and transmit them through to the endpoint. If the mode was set to “inactive”, the gateway MUST (as always) discard the voice signals received on the connection. If the mode was set to “network loopback” or “network continuity test” the gateway MUST perform the expected echo or response. The echoed or generated media MUST then be sent to the source of the media received. Note, that when the endpoint does not have a RemoteConnectionDescriptor for the connection, the connection can by definition not be in any of the modes “sendonly”, “send/receive”, “replicate”, or “conference”.

The following parameters are optional and they can be used by the Call Agent to include a NotificationRequest which will execute simultaneously with the CreateConnection command:

- RequestedEvents,
- RequestIdentifier,
- DigitMap,
- SignalRequests,
- QuarantineHandling,
- DetectEvents

If one or more of these parameters is present, then the RequestIdentifier must also be present. Thus, the inclusion of a notification request can be recognized by the presence of a RequestIdentifier. The rest of the parameters may or may not be present. If one of the parameters is not present, it must be treated as if it was a normal NotificationRequest with the parameter in question being omitted. This may have the effect of canceling signals and of stop looking for events.

EXAMPLE

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

ask the embedded client to create a connection, in order to be sure that the user can start speaking as soon as the phone goes off hook,

ask the embedded client to start ringing,
ask the embedded client to notify the Call Agent when the phone goes off-hook.

All of the above can be accomplished in a single CreateConnection command by including a notification request with the RequestedEvents parameters for the off-hook event and the SignalRequests parameter for the ringing signal. When these parameters are present, the creation of the connection and the notification request MUST be synchronized, which means that they are both either accepted or refused. In our example, the CreateConnection must be refused if the gateway does not have sufficient resources or cannot get adequate resources from the local network access. The off-hook notification request must be refused in the glare condition if the user is already off-hook. In this example, the phone must not ring if the connection cannot be established, and the connection must not be established if the user is already off-hook. An error would be returned instead (error code 401 – phone off hook), which informs the Call Agent of the glare condition.

`ReturnCode`

This is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

3.4.4 ModifyConnection

This command is used to modify the characteristics of an already established connection.

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

`ConnectionId`

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.

Endpoint

The EndpointId MUST be a fully qualified endpoint name.

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 **LocalConnection-Options**, 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. 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. The RTP address information provided in the RemoteConnectionDescriptor specifies the remote RTP address of the receiver of media for the connection. The Call Agent may have changed this RTP address information. When RTP address information is given to an embedded client for a connection, the embedded client SHOULD only accept media streams (and RTCP) from the RTP address specified as well. Any media streams received from any other addresses SHOULD be discarded.

The **RequestedEvents, RequestIdentifier, DigitMap, Signal-Requests, QuarantineHandling**, and **DetectEvents** parameters are optional. The parameters can be used by the Call Agent to include a notification request that is tied to and executed simultaneously with the connection modification. If one or more of these parameters is supplied, then RequestIdentifier MUST be one of them.

ReturnCode

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

3.4.5 DeleteConnection

Both the Call Agent and Embedded Clients can send the DeleteConnection command.

3.4.5.1 DeleteConnection from the Call Agent

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

ReturnCode
, Connection-parameters
← DeleteConnection (CallId
, EndpointId
, ConnectionId
[, NotifiedEntity]
[, RequestedEvents]
[, RequestIdentifier]
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

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.

`EndpointId`

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

`CallId and ConnectionId`

These are the identities of the call and connection to be deleted.

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

ReturnCode

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

Connection-Parameters

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

- *Number of packets sent:* The total number of RTP data packets transmitted by the sender since starting transmission on the connection. The count is not reset if the sender changes its synchronization source identifier (SSRC, as defined in RTP)for example, as a result of a Modify command. The value is zero if, e.g., the connection was always set in “receive only” mode.

- *Number of octets sent:* The total number of payload octets (i.e., not includingheader or padding) transmitted in RTP data packets by the sender since starting transmission on the connection. The count is not reset if the sender changes its SSRC identifier—for example, as a result of a ModifyConnection command. The value is zero if, e.g., the connection was always set in “receive only” mode.

- *Number of packets received:* The total number of RTP data packets received by the sender since starting reception on the connection. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always set in “send only” mode.

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

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

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

- *Average transmission delay:* An estimate of the network latency, expressed in milliseconds. This is the average value of the difference between the NTP timestamp indicated by the senders of the RTCP messages and the NTP timestamp of the receivers, measured when the messages are received. The average is obtained by summing all the estimates and then dividing by the number of RTCP messages that have been received. It should be noted that the correct calculation of this parameter relies on synchronized clocks. Embedded client devices MAY alternatively estimate the average transmission delay by dividing the measured roundtrip time by two.

3.4.5.2 DeleteConnection from the Embedded Client

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

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

`Endpoint`

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

`Reason-Code`

It is a text string starting with a numeric reason-code and optionally followed by a descriptive text string. The reason code indicates the cause of the DeleteConnection.

In addition to the CallId, EndpointId, and ConnectionId, the embedded client will also send the connection’s parameters, which would have been returned to the Call Agent in response to a DeleteConnection command from the Call Agent..

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

3.4.5.3 Delete Multiple Connections From the Call Agent

A variation of the DeleteConnection function can be used by the Call Agent to delete multiple connections at the same time. The command can be used to delete all connections that relate to a call for an endpoint:

ReturnCode
← DeleteConnection (CallId,
EndpointId)

The **EndpointId**, in this form of the DeleteConnection command, MUST NOT use the "any of" wildcard. All connections for the endpoint(s) with the CallId specified will be deleted. The command does not return any individual statistics or call parameters.

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

ReturnCode
← (EndpointId)

In this form of the DeleteConnection command, Call Agents can take advantage of the hierarchical naming structure of endpoints to delete all the connections that belong to a group of endpoints. In this case, part of the "local endpoint name" component of the EndpointId can be specified using the "all" wildcarding convention, as specified in Section 4.1.1. The "any of" wildcarding convention MUST NOT be used. The command does not return any individual statistics or call parameters. After the connection has been deleted, packet network media streams previously supported by the connection are no longer available. Any media packets received for the old connection are simply discarded and no new media packets for the stream are sent.

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

3.4.6 AuditEndpoint

This command can be used by the Call Agent to detect the status of a particular endpoint.

{ ReturnCode

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

`EndpointId`

This identifies the endpoint that is being audited. The “any of” wildcard convention must not be used. The “all of” wildcard convention can be used to audit a group of endpoints. If thisconvention is used, the gateway must return the list of endpoint identifiers that match the wildcard in the *EndPointIdList* parameter, which is simply a list of *SpecificEndpointIds. RequestedInfo* must not be included in this case.

`MaxEndPointIDs`

This is a numerical value that indicates the maximum number of EndpointIds to return. If additional endpoints exist, the *NumEndPoints* return parameter must be present and indicate the total number of endpoints that match the EndpointID specified.

In order to retrieve the next block of EndpointIDs, the *SpecificEndPointID* is set to the value of the last endpoint returned in the previous *EndPointIDList,* and the command is issued.

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

- RequestedEvents ,
- DigitMap ,
- SignalRequests ,
- RequestIdentifier ,
- NotifiedEntity ,

- ConnectionIdentifiers ,
- DetectEvents ,
- ObservedEvents ,
- EventStates ,
- VersionSupported ,
- Capabilities

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

`RequestedEvents`

This is the current value of RequestedEvents the endpoint is using including the action associated with each event. Persistent events are included in the list.

`DigitMap`

This is the digit map currently in use by the endpoint.

`SignalRequests`
This is a list of currently active Time-Out signals, pending Brief signals, and any On-Off signals that are currently “on” for the endpoint. Parameters for signals are also included.

`RequestIdentifier`

This is the RequestIdentifier for the last NotificationRequest received by the endpoint (including those sent with CreateConnection or ModifyConnection commands), or zero if none has been received.

`NotifiedEntity`

The current “notified entity” for the endpoint.

`ConnectionIdentifiers`

A comma-separated list of ConnectionIdentifiers for all connections that currently exist for the specified endpoint.

`DetectEvents`

The current value of DetectEvents the endpoint is using. Persistent events are included in the list.

`ObservedEvents`

The current list of observed events for the endpoint.

EventStates

This is for events that have auditable states. The event name is given according to the state of the endpoint e.g. off-hook if the endpoint is in the off-hook state.

VersionSupported

A list of protocol versions supported by the endpoint.

Capabilities

The capabilities for the endpoint are similar to the LocalConnectionOptions parameter and include event packages and connection modes. If there is a need to specify that some parameters, such as e.g., silence suppression, are only compatible with some codecs, then the gateway will return several capability sets.

- *Compression Algorithm*

A list of supported codecs. The rest of the parameters will apply to all codecs specified in this list.

- *Packetization Period*

A single value or a range may be specified.

- *Bandwidth*

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

- *Echo Cancellation*

Whether echo cancellation is supported or not.

- *Silence Suppression*

Whether silence suppression is supported or not.

- *Type of Service*

Whether type of service is supported or not.

- *Event Packages*

A list of event packages supported. The first event package in the list will be the default package.

Modes

A list of supported connection modes.

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

ReturnCode

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

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

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

3.4.7 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])

EndpointId

It identifies the endpoint that is being audited—wildcards must not be used.

RequestedInfo and ConnectionId

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

- CallId,
- NotifiedEntity,
- LocalConnectionOptions,
- Mode,
- ConnectionParameters,

- RemoteConnectionDescriptor,
- LocalConnectionDescriptor

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

- **CallId**
-

The CallId for the call to which the connection belongs.

- **NotifiedEntity**

The current “notified entity” for the endpoint.

- **LocalConnectionOptions**

The LocalConnectionOptions supplied for the connection.

- **Mode**

The current connection mode.

- **ConnectionParameters**

Current connection parameters for the connection.

- **LocalConnectionDescriptor**

The LocalConnectionDescriptor that the gateway supplied for the connection.

- **RemoteConnectionDescriptor**

The RemoteConnectionDescriptor that was supplied to the gateway for the connection.

- **ReturnCode**

This is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number 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).

3.4.5 Restart In Progress

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

Return code
[,NotifiedEntity]
[,VersionSupported]
←RestartInProgress(EndpointId
, RestartMethod
[, RestartDelay])

`EndpointId`

The EndpointId identifies the endpoints that are taken in or out of service. The "all of" wildcard convention can be used to apply the command to a group of endpoints,for example, all endpoints that are attached to a specified interface, or even all endpoints that are attached to a given gateway.

`RestartMethod`

The RestartMethod parameter specifies the type of restart:

- A ***"graceful"*** restart method indicates that the specified endpoint(s) will be taken out of service after the specified "restart delay". The established connections are not yet affected, but the Call Agent should refrain from establishing new connections, and should try to gracefully tear down any existing connections.
- A ***"forced"*** restart method indicates that the specified endpoints are taken out of service abruptly. The established connections, if any, are lost.

- A "***restart***" method indicates that service will be restored on the endpoints after the specified "restart delay". There are no connections that are currently established on the endpoints.
- A "***disconnected"*** method indicates that the endpoint has become disconnected and is now trying to establish connectivity. The "restart delay" specifies the number of seconds the endpoint has been disconnected. Established connections are not affected.

`RestartDelay`

The optional "restart delay" parameter is expressed as a number of seconds. If the number is absent, the delay value should be considered null. In the case of the "graceful" method, a null delay indicates that the Call Agent should simply wait for the natural termination of the existing connections, without establishing new connections. The restart delay is always considered null in the case of the "forced" method. A restart delay of null for the "restart" method indicates that service has already been restored. This typically will occur after gateway startup/reboot message

as a courtesy to the Call Agent when they are taken out of service, e.g., by being shutdown, or taken out of service by a network management system, although the Call Agent cannot rely on always receiving such messages. Embedded clients MUST send a “restart” RestartInProgress message with a null delay to their Call Agent when they are back in service according to the restart procedure specified in Section 3.5.3.3 – Call Agents can rely on receiving this message. Also, embedded clients MUST send a “disconnected” restartInProgress message to their current “notified entity” according to the “disconnected” procedure specified in Section 3.5.2/3.5.3.4. The “restart delay” parameter MUST NOT be used with the “forced” restart method.The RestartInProgress message will be sent to the current “notified entity” for the EndpointId in question. It is expected that a default Call Agent, i.e., “notified entity”,has been provisioned for each endpoint so, after a reboot, the default Call Agent will be the “notified entity” for each endpoint. Embedded clients MUST take full advantage of wild-carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent.

`ReturnCode`

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

`NotifiedEntity`

A Notified Entity may additionally be returned with the response from the Call Agent:

- If the response indicated success (return code 200 – transaction executed), the restart procedure has completed, and the NotifiedEntity returned is the new “notified entity” for the endpoint(s).
- If the response from the Call Agent indicated an error, the restart procedure is not yet complete, and must therefore be initiated again, and the NotifiedEntity returned specifies the new “notified entity” for the endpoint(s) which should be used when retrying.

VersionSupported

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

3.5 STATES, FAILOVER AND RACE CONDITIONS

In order to implement proper call signaling, the Call Agent must keep track of the state of the endpoint, and the gateway must make sure that events are properly notified to the call agent. Special conditions may exist when the gateway or the call agent are estarted: the gateway may need to be redirected to a new call agent

during “failover” procedures; Similarly, the call agent may need to take special action when the gateway is taken offline, or restarted.

3.5.1 Recaps and Highlights

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

- A Call Agent is identified by its domain name, not its network addresses, and several network addresses can be associated with a domain name.

- An endpoint has one, and only one, Call Agent associated with it at any given point in time. The Call Agent associated with an endpoint is the current value of the “notified entity”.

- The “notified entity” is initially set to a provisioned value. When commands with a NotifiedEntity parameter is received for the endpoint, including wild-carded endpoint-names, the “notified entity” is set to the value specified. If the “notified entity” for an endpoint is empty or has not been set explicitly[1], the “notified entity” defaults to the source address of the last connection handling command or notification request received for the endpoint. In this case, the Call Agent will thus be identified by its network address, which SHOULD only be done on exceptional basis.

- Responses to commands are always sent to the source address of the command, regardless of the current “notified entity”. When a Notify message needs to be piggy-backed with the response, the datagram is still sent to the source address of the new command received, regardless of the NotifiedEntity for any of the commands.

- When the “notified entity” refers to a domain name that resolves to multiple IP-addresses, endpoints are capable of switching between each of these addresses, however they cannot change the “notified entity” to another domain name on their own. A call agent can however instruct them to switch by providing them with a new “notified entity”.
-
- If a call agent becomes unavailable, the endpoints managed by that call agent will eventually become “disconnected”. The only way for these endpoints to become connected again is either for the failed call agent to become available again, or for another (backup) call agent to contact the affected endpoints with a new “notified entity”.

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

- When another (backup) call agent has taken over control of a group of endpoints, it is assumed that the failed call agent will communicate and synchronize with the backup call agent in order to transfer control of the affected endpoints back to the original call agent, if so desired. Alternatively, the failed call agent could simply become the backup call agent now.

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

3.5.2 Retransmission, and Detection of Lost Associations

The MGCP/NCS protocol is organized as a set of transactions, each of which is composed of a command and a response. The MGCP/NCS messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are repeated. Gateways MUST keep in memory a list of the responses that they sent to recent transactions, and a list of the transactions that are currently being executed. Recent is here defined by the value Tthist that specifies the number of seconds that responses to old transactions must be kept for. The default value for Tthist is 30 seconds.
The transaction identifiers of incoming commands are first compared to the transaction identifiers of the recent responses. If a match is found, the gateway does not execute the transaction, but simply repeats the old response. If a match to a previously responded to transaction is not found, the transaction identifier of the incoming command is compared to the list of transactions that have not yet finished executing. If a match is found, the gateway does not execute the transaction, which is simply ignored - a response will be provided when the execution of the command is complete. This repetition mechanism is used to guard against four types of possible errors:

- transmission errors, when, e.g., a packet is lost due to noise on a line or congestion in a queue,
- component failure, when, e.g., an interface for a call agent becomes unavailable,
- call agent failure, when, e.g., all interfaces for a call agent becomes unavailable,
- failover, when a new call agent is “taking over” transparently.

The elements should be able to derive from the past history an estimate of the packet loss rate. In a properly configured system, this loss rate should be very low, typically less than 1% on average. If a call agent or a gateway has to repeat a message more than a few times, it is very legitimate to assume that something else than a transmission error is occurring. For example, given a uniformly distributed loss rate of 1%, the probability that 5 consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a call agent that processes 1,000 transactions per second. (Indeed, the number of repetitions

that is considered excessive should be a function of the prevailing packet loss rate.) When errors are non-uniformly distributed, the consecutive failure probability can become somewhat higher. We should note that the “suspicion threshold”, which we will call “Max1”, is normally lower than the “disconnection threshold”, which we will call “Max2”, and which should be set to a larger value.

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

- The gateway MUST always check for the presence of a new call agent. It can be noticed by:

- receiving a command where the **NotifiedEntity** points to a new call agent , or receiving a redirection response pointing to a new call agent .

- If a new Call Agent is detected, the gateway MUST direct retransmissions of any outstanding commands for the endpoint(s) redirected to that new Call Agent. Responses to new or old commands are still transmitted to the source address of the command.

- Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than Tsmax. If more than Tsmax time has elapsed, the endpoint becomes disconnected.

- If the number of retransmissions to this Call Agent equals “Max1”, the gateway MAY actively query the name server in order to detect the possible change of call agent interfaces, regardless of the Time To Live (TTL) associated with the DNS record.

- The gateway may have learned several IP addresses for the Call Agent. If the number of retransmissions for this IP address is larger than “Max1” and lower than “Max2", and there are more IP addresses that have not been tried, then the gateway MUST direct the retransmissions to the remaining alternate addresses in its local list.
- If there are no more interfaces to try, and the number of retransmissions is Max2, then the gateway SHOULD contact the DNS one more time to see if any other interfaces have become available. If not, the endpoint(s) managed by this Call Agent are now disconnected. When an endpoint becomes disconnected, it MUST then initiate the “disconnected” procedure .

In order to adapt to network load automatically, MGCP/NCS specifies exponentially increasing timers (see Section 0). If the initial time-out is set to 200 milliseconds, the loss of a fifth retransmission will be detected after about 6 seconds. This is probably an acceptable waiting delay to detect a failover. The retransmissions should

continue after that delay not only in order to perhaps overcome a transient connectivity problem, but also in order to allow some more time for the execution of a failover - waiting a total delay of 30 seconds is probably acceptable.
It should be noted, that there is an intimate relationship between Tsmax, Tthist, and the maximum transit time, Tpmax. Specifically, the following relation MUST be satisfied to prevent retransmitted commands from being executed more than once:

$Tt_{hist} >= Ts_{max} + Tp_{max}$

The default value for Tsmax is 20 seconds. Thus, if the assumed maximum propagation delay is 10 seconds, then responses to old transactions must be kept for a period of at least 30 seconds. The importance of having the sender and receiver agree on these values cannot be overstated.

The default value for Max1 is 5 retransmissions and the default value for Max2 is 7 retransmissions. Both of these values may be altered by the provisioning process. Furthermore, the provisioning process MUST be able to disable one or both of the Max1 and Max2 DNS queries.

3.5.3 Race Conditions

3.5.3.1 Explicit Detection

A key element of the state of several endpoints is the position of the hook. Although hook-state changing events are persistent in NCS, race conditions and state mismatch may still occur. For example when the user decides to go off-hook while the Call Agent is in the process of requesting the gateway to look for off-hook events and perhaps apply a ringing signal (the “glare” condition well known in voice-based capabilities).
To avoid this race condition, the gateway MUST check the condition of the endpoint before responding to a NotificationRequest. Specifically, it MUST return an error:

1. If the gateway is requested to notify an “off hook” transition while the phone is already off hook (error code 401 – phone off hook),

2. If the gateway is requested to notify an “on hook” or “flash hook” condition while the phone is already on hook (error code 402 – phone on hook).

Additionally, individual signal definitions can specify that a signal will only operate under certain conditions, e.g., ringing may only be possible if the phone is already off hook. If such prerequisites exist for a given signal, the gateway MUST return the error specified in the signal definition if the prerequisite is not met.
It should be noted, that the condition check is performed at the time the notification request is received, where as the actual event that caused the current condition may have either been reported, or ignored earlier, or it may currently be quarantined.
The other state variables of the gateway, such as the list of requested events or list of requested signals, are entirely replaced after each successful NotificationRequest,

which prevents any long term discrepancy between the Call Agent and the gateway. When a NotificationRequest is unsuccessful, whether it is included in a connection-handling command or not, the gateway will simply continue as if the command had never been received. although an error is returned. As all other transactions, the NotificationRequest MUST operate as an atomic transaction, Thus any changes initiated as a result of the command MUST be reverted.
Another race condition can occur when a Notify is issued shortly before the reception by the gateway of a NotificationRequest. The RequestIdentifier is used to correlate Notify commands with NotificationRequest commands thereby enabling the Call Agent to determine if the Notify command was generated before or after the gateway received the new NotificationRequest.

3.5.3.2 Ordering of Commands, and Treatment of Disorder

NCS does not mandate that the underlying transport protocol guarantees the sequencing of commands sent to a gateway or an endpoint. This property tends to maximize the timeliness of actions, but it has a few drawbacks. For example:

- Notify commands may be delayed and arrive to the call agent after the transmission of a new Notification Request command,

- If a new NotificationRequest is transmitted before a response to a previous one is received, there is no guarantee that the previous one will not be received in second position.

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

1. When a gateway handles several endpoints, commands pertaining to the different endpoints can be sent in parallel, for example following a model where each endpoint is controlled by its own process or its own thread.

2. When several connections are created on the same endpoint, commands pertaining to different connections can be sent in parallel.

3. On a given connection, there should normally be only one outstanding command (create or modify). However, a DeleteConnection command can be issued at any time. In consequence, a gateway may sometimes receive a ModifyConnection command that applies to a previously deleted connection. Such commands MUST be ignored, and an error returned (error code 515 – incorrect connection-id).

4. On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter is used to correlate Notify commands with the triggering NotificationRequest.

5. In some cases, an implicitly or explicitly wild-carded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The Call Agent should individually delete all connections whose

completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wild-carding should not be sent until a response to the wild-carded DeleteConnection command is received.

6. When commands are embedded within each other, sequencing requirements for all commands MUST be adhered to. For example a CreateConnection command with a notification request in it must adhere to the sequencing requirements for CreateConnection and NotificationRequest at the same time.

7. AuditEndpoint and AuditConnection are not subject to any sequencing.

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

9. When multiple messages are piggy-backed in a single packet, the messages are always processed in order.

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

3.5.3.3 Fighting the Restart Avalanche

Let's suppose that a large number of gateways are powered on simultaneously. If they were to all initiate a RestartInProgress transaction, the Call Agent would very likely be swamped, leading to message losses and network congestion during the critical period of service restoration. In order to prevent such avalanches, the following behavior MUST be followed:

1. When a gateway is powered on, it initiates a restart timer to a random value, uniformly distributed between 0 and a provisionable maximum waiting delay (MWD), e.g., 360 seconds (see below). Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways that would use the same algorithm.

2. The gateway then waits for either the end of this timer, the reception of a command from the call agent, or the detection of a local user activity, such as for example an off-hook transition on a residential gateway. A pre-existing off-hook condition results in the generation of an off-hook event.

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

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

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

It is expected that each endpoint in a gateway will have a provisionable Call Agent, i.e., “notified entity”, to direct the initial restart message towards. When the collection of endpoints in a gateway is managed by more than one Call Agent, the above procedure must be performed for each collection of endpoints managed by a given Call Agent. The gateway MUST take full advantage of wild-carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent.

The value of MWD is a configuration parameter that depends on the type of the gateway. The following reasoning can be used to determine the value of this delay on residential gateways.

Call agents are typically dimensioned to handle the peak hour traffic load, during which, on average, 10% of the lines will be busy, placing calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 transactions between each endpoint and the Call Agent. This simple calculation shows that the Call Agent is expected to handle 5 to 6 transactions for each endpoint, every 30 minutes on average, or, to put it otherwise, about one transaction per endpoint every 5 to 6 minutes on average. This suggests that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, embedded clients MUST use a default value of 600 seconds for MWD.

3.5.3.4 Disconnected Endpoints

In addition to the restart procedure, embedded clients also have a “disconnected” procedure, which is initiated when an endpoint becomes “disconnected” as described in Section 4.4.2. It should here be noted, that endpoints can only become disconnected when they attempt to communicate with the Call Agent. The following steps are followed by an endpoint that becomes “disconnected”:

1. A “disconnected” timer is initialized to a random value, uniformly distributed between 0 and a provisionable “disconnected” initial waiting delay (Td_{init}), e.g., 15 seconds. Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways and endpoints that would use the same algorithm.

2. The gateway then waits for either the end of this timer, the reception of a command from the call agent, or the detection of a local user activity for the endpoint, such as for example an off-hook transition.

3. When the “disconnected” timer elapses, when a command is received, or when a local user activity is detected, the gateway initiates the “disconnected” procedure for the endpoint. In the case of local user activity, a provisionable “disconnected” minimum waiting delay (Td_{min}) must furthermore have elapsed since the gateway became disconnected or the last time it initiated the “disconnected” procedure in order to limit the rate at which the procedure is performed.

4. If the “disconnected” procedure still left the endpoint disconnected, the “disconnected” timer is then doubled, subject to a provisionable “disconnected” maximum waiting delay (Td_{max}), e.g., 600 seconds, and the gateway proceeds with step 2 again.

The “disconnected” procedure is similar to the restart procedure. In that it simply states that the endpoint MUST send a RestartInProgress command to the Call Agent informing it that the endpoint was disconnected and furthermore guarantee that the first message (command or response) that the Call Agent now sees from this endpoint MUST be this RestartInProgress command. The endpoint MUST take full advantage of piggy-backing in achieving this. The Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint.
This specification purposely does not specify any additional behavior for a disconnected endpoint. Vendors MAY for instance choose to provide silence, play reorder tone, or even enable a downloaded wav file to be played 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.

3.6 THE FORMAT OF NCS COMMANDS AND RESPONSES

The NCS implements its control functions as a set of transactions. The transactions are composed of a command and a mandatory response. There are eight types of commands:

- CreateConnection
- ModifyConnection
- DeleteConnection
- NotificationRequest
- Notify

• AuditEndpoint
• AuditConnection
• RestartInProgress

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

The mandatory response is of two types. Either the response is a simple acknowledgement of the command received by the gateway, also acknowledging normal and successful execution of the command, or it is an acknowledgment plus some information requested by the Call Agent itself. In the latter case, the response can be termed as “response with information”.

3.6.1 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.
NCS 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.

3.6.2 Command Header Format

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.

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.

3.6.2.1 Command Line

The command line is composed of:

1. The name of the requested verb,

2. The identification of the transaction,

3. 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),

4. The protocol version.

These four items are encoded as strings of printable ASCII characters separated by white spaces, i.e., the ASCII space (0x20). Embedded clients should use exactly one ASCII space separator, however they must be able to parse messages with additional white space characters.

3.6.2.1.1 Requested Verb Coding

Requested verbs are encoded as four letter upper- and/or lower-case ASCII codes as defined in the following table:

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

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

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

3.6.2.1.2 Transaction Identifiers

Transaction identifiers are used to correlate commands and responses.

Transaction identifiers for commands sent to a given embedded client must be unique for maximum lifetime of the transactions because the embedded clients can always detect duplicate transactions by simply examining the transaction identifier.

Transaction identifiers for all commands sent from a given embedded client must also be unique for maximum lifetime of the transactions. Thus, a Call Agent can always detect a duplicate transaction from an embedded client by the combination of the domain-name of the endpoint and the transaction identifier. The embedded client in turn can always detect a duplicate response acknowledgement by looking at the transaction id(s).

The transaction identifier is encoded as a string of up to nine decimal digits. In the command lines, it immediately follows the coding of the verb. Transaction identifiers have values between 1 and 999999999. 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.

3.6.2.1.3 Endpoint, Call Agent and NotifiedEntity Name Coding

The endpoint names and Call Agent names are encoded as defined in RFC 821,for example,
`Local-name@domain-name`

In these addresses, the domain name identifies the system where the endpoint is attached, while the left side identifies a specific endpoint on that system. Both components must be case insensitive.

Examples of such names are:

`aaln/1@ncs2.whatever.net`

Analog access line 1 in the embedded client ncs2 in the “Whatever” network.

`Call-agent@ca.whatever.net`

Call Agent for the “whatever” network.

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

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

In case the port number is omitted, the default MGCP Call Agent port (2727, unless provisioned otherwise) will be used. Additional detail on endpoint names can be found in chapter 1

3.6.2.1.4 Protocol Version Coding

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

```
MGCP 1.0 NCS 1.0.
```

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

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

3.6.2.2 Parameter Lines

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

Parameter Name	Code
Response Acknowledgement	K
CallId	C
ConnectionId	I
NotifiedEntity	N
RequestIdentifier	X
LocalConnectionOptions	L
Connection Mode	M
RequestedEvents	R
SiqnalRequests	S
DigitMap	D
ObservedEvents	O
ConnectionParameters	P
ReasonCode	E
SpecificEndpointId	Z
MaxEndpointIds	ZM
NumEndpoints	ZN

RequestedInfo	F
QuarantineHandling	Q
DetectEvents	T
EventStates	ES
RestartMethod	RM
ResartDelay	RD
Capabilities	A
VersionSupported	VS

These parameters are not necessarily present in all the commands.The following table tells which parameters are allowed in which commands. The letter M stands for Mandatory, O for Optional and F for Forbidden.

ParameterName	**CRCX**	**MDCX**	**DLCX**	**RQNT**	**NTFY**	**AUEP**	**AUCX**	**RSIP**
ResponseAck	O	O	O	O	O	O	O	O
CallId	M	M	O	F	F	F	F	F
ConnectionId	F	M	O	F	F	F	M	F
RequestIdentifier	O	O	O	M	M	F	F	F
LocalConnectionOptions	M	O	F	F	F	F	F	F
Connection Mode	M	O	F	F	F	F	F	F
RrequestedInfo	O	O	O	O	F	F	F	F
SignalRequests	O	O	O	O	F	F	F	F
NotifiedEntity	O	O	O	O	O	F	F	F
ReasonCode	F	F	O	F	F	F	F	F
ObservedEvents	F	F	F	F	M	F	F	F
DigitMap	O	O	O	O	F	F	F	F
ConnectionParameters	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
VersionSuported	F	F	F	F	F	F	F	F
RemoteConnectionDescriptor	O	O	F	F	F	F	F	F

Embedded clients and Call Agents SHOULD always provide mandatory parameters

before optional ones, however embedded clients MUST NOT fail if this recommendation is not followed.

3.6.2.2.1 Response Acknowledgement

The response acknowledgement parameter is used to support the three-way handshake . It contains a comma separated list of "confirmed transaction-id ranges". Each "confirmed transaction-id range" is composed of either one decimal number, when the range includes exactly one transaction, or two decimal numbers separated by a single hyphen, describing the lower and higher transaction identifiers included in the range. An example of a response acknowledgement is:

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

3.6.2.2.2 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest that triggered it. A RequestIdentifier is a hexadecimal string not more than 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 . For example:

```
X: 7324778
```

3.6.2.2.3 Local Connection Options

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

- The packetization period in milliseconds, encoded as the keyword “p” followed by a colon and a decimal number.
- The literal name of the compression algorithm as specified in the Packet Cable Audio/Video Codec Specification, PKT-SP-D05-990319, Section 5.4 (Table 2), encoded as the keyword “a” followed by a colon and a character string. These literal names are equivalent to the codec definitions in RTP Parameters [19]. 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”.

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

Examples of local connection options are:

```
L: p:10, a:PCMU
L: p:10, a:PCMU, e:off, t:20, s:on
L: p:30, a:G729A, e:on, t:A0, s:off
```

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

3.6.2.2.4 Capabilities

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

The parameters used are:

- The packetization period in milliseconds, encoded as the keyword “p” followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen.

- The literal name of the compression algorithm, encoded as the keyword “a” followed by a colon and a character string. A list of values may be specified in which case the values will be separated by a semicolon.

- The bandwidth in kilobits per second (1000 bits per second), encoded as the keyword “b” followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen.

- The echo-cancellation parameter, encoded as the keyword “e” followed by a colon and the value “on” if echo cancellation is supported, “off” otherwise.

- The type of service parameter, encoded as the keyword “t” followed by a colon and the value “0” if type of service is not supported, all other values indicate support for type of service.

- The silence suppression parameter, encoded as the keyword “s” followed by a colon and the value “on” if silence suppression is supported, “off” otherwise.

- The event packages supported by this endpoint encoded as the keyword “v” followed by a colon and then a semicolon-separated list of package names supported. The first value specified will be the default package for the endpoint.

- The connection modes supported by this endpoint encoded as the keyword “m” followed by a colon and a semicolon-separated list of connection modes supported as defined in Section 5.2.2.7.

When several parameters are present, the values are separated by a comma. Examples of capabilities are:

```
A: a:PCMU;G729A, p:10-100, e:on, s:off, v:L;S,
   m:sendonly;recvonly;sendrecv;inactive

A: a:G729A; p:30-90, e:on, s:on, v:L;S,
   m:sendonly;recvonly;sendrecv;inactive;confrnce
```

3.6.2.2.5 Connection Parameters

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

The connection parameter types are specified in the following table:

Connection Parameter Name	**Code**	**Connection Parameter Value**
Packets sent	PS	The number of packets that were sent on the connection.
Octets sent	OS	The number of octets that were sent on the connection.
Packets received	PR	The number of packets that were received on the connection.
Octets received	OR	The number of octets that were received on the connection.
Packets lost	PL	The number of packets that were not received on the connection, as deduced from gaps in the sequence number.
Jitter	JI	The average inter-packet arrival jitter, in milliseconds, expressed as an integer number.
Average Latency	LA	Average latency, in milliseconds, expressed as an integer number.

An example of a connection parameter encoding is:

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

3.6.2.2.6 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 Appendix C.

3.6.2.2.7 Connection Mode

The connection mode describes the connection's operation mode. The possible values are:

Connection Mode	**Meaning**
M: sendonly	The gateway should only send packets.
M: recvonly	The gateway should only receive packets.
M: sendrecv	The gateway should send and receive packets.
M: confrnce	The gateway should send and receive packets according to conference mode.
M: inactive	The gateway should neither send nor receive packets.
M: replcate	The gateway should only send packets according to replicate mode..
M: netwloop	The gateway should place the endpoint in Network Loopback mode.
M: netwtest	The gateway should place the endpoint in Network Continuity Test mode.

3.6.2.2.8 Event/Signal Name Coding

Event/signal names are composed of an optional package name, separated by a slash (/) from the name of the actual event. The event name can optionally be followed by an at sign (@) and the identifier of a connection on which the event should be observed. Event names are used in the RequestedEvents, SignalRequests, DetectEvents, ObservedEvents, and EventStates parameters. Each event is identified by an event code. These ASCII encodings are not case sensitive. Values such as "hu", "Hu", "HU" or "hU" should be considered equal.
The following are valid examples of event names:

L/hu	On-hook transition in Line package
L/0	Digit 0 in Line package

hf	Flash-hook
L/rt@98593	Ringback on connection ”98593”

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

L/[0-9]	Digits 0 to 9 in Line package
L/X	Digit 0 to 9 in Line package
[0-9*#A-D]	All digits and letters in Line package
L/all	All events in Line package

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

L/ma@*	The RTP media strt event on all connections
L/rt@$	Ringback on current connection

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

3.6.2.2.9 RequestedEvents

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

Each event can be qualified by a requested action, or by a list of actions. Not all actions can be combined – please refer to Section 4.3.1 for valid combinations. The actions, when specified, are encoded as a list of keywords enclosed in parenthesis and separated by commas. The codes for the various actions are:

Action	Code
Notify immediately	N
Accumulate	A
Accumulate according to digit map	D
Ignore	I
Keep signal active	K
Embedded Notification Request	E
Embedded Modify connection	C

If a digit map is not provided when the “accumulate according to digit map” action is

specified, the endpoint simply uses its current digit map. If the endpoint does not have any digit maps currently, an error MUST be returned (error code 519 – no digit map).

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

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

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

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

The embedded NotificationRequest follows the format:

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

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

```
R:hd(A,E(S(dl),R(oc(N),[0-9#T](D)),D((1xxxxxxxxxx|9011x.T))) )
```

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

The embedded ModifyConnection action follows the format:

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

The following example illustrates the use of Embedded ModifyConnection:

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

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

3.6.2.2.10 SignalRequests

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

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

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

where signal-parameter-value may be either a string or a quoted string, i.e., a string surrounded by two double quotes. Two consecutive double-quotes in a quoted string will escape a double-quote within that quoted string. For example, "ab""c" will produce the string ab"c.
Each signal has one of the following signal-types associated with it (see section 4.3.1, p. 17):

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

On/Off signals can be parameterized with a "+" to turn the signal on, or a "-" to turn the signal off. If an on/off signal is not parameterized, the signal is turned on. Both of the following will turn the vmwi signal on:
vmwi(+), vmwi
Time-out signals can be parameterized with the signal parameter "TO" and a time-out value that overrides the default time-out value. If a time-out signal is not parameterized with a time-out value the default time-out value will be used. Both of the following will apply the ringing signal for 6 seconds:

```
rg(to=6000)
rg(to(6000))
```

Individual signals may define additional signal parameters.
The signal parameters will be enclosed within parenthesis, as in (assuming "Line" is the default package):

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

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

```
S: rg, rt@FDE243CD
```

3.6.2.2.11 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: hu
O: ma@A43B81
O: 8,2,9,5,5,5,5,T
O: hf,hf,hu
O: 8,2,9,5,mt,5,5,5,T
```

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

3.6.2.2.12 RequestedInfo

The RequestedInfo parameter contains a comma-separated list of parameter codes, as defined in the "Parameter lines" section 3.6.2.2 .Following lists the parameters that can be audited. The following values are supported as well:

LocalConnectionDescriptor	LC
RemoteConnectionDescriptor	RC

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

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

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

```
F: A
```

3.6.2.2.13 QuarantineHandling

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

```
Q: process
```

3.6.2.2.14 DetectEvents

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

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

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

3.6.2.2.15 EventStates

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

```
ES: hu
```

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

3.6.2.2.16 RestartMethod

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

```
RM: restart
```

3.6.2.2.17 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 NCS 1.0
```

3.6.3 Response Header Formats

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

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

```
200 1201 OK
```

The following table summarizes the response parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response assuming the command succeeded. The reader should still

study the individual command definitions though as this table only provides summary information. The letter M stands for mandatory, O for optional, and F for forbidden.

ParameterName	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ResponseAck	O	O	O	O	O	O	O	O
CallId	F	F	F	F	F	F	O	F
ConnectionId	M	F	F	F	F	O	F	F
RequestIdentifier	F	F	F	F	F	O	F	F
LocalConnectionOptions	F	F	F	F	F	O	O	F
Connection Mode	F	F	F	F	F	F	O	F
RrequestedInfo	F	F	F	F	F	O	F	F
SignalRequests	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	O	O	O
ReasonCode	F	F	F	F	F	F	F	F
ObservedEvents	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	O	F	F
ConnectionParameters	F	F	O	F	F	F	O	F
SpecificEndpointId	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
Capablilities	F	F	F	F	F	O	F	F
LocalConnectionOptions	M	O	F	F	F	F	O	F
RemoteConnectionDescriptor	F	F	F	F	F	F	O	F

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

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

3.6.3.1 CreateConnection

In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter with a successful response (code 200). Local ConnectionDescriptor is furthermore transmitted with a positive response. The

LocalConnectionDescriptor is encoded as a “session description”, as defined in section It is separated from the response header by an empty line, e.g.:

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96
```

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

```
200 1204 OK
K:
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96
```

The final response is acknowledged by a Response Acknowledgement:

```
000 1204
```

3.6.3.2 ModifyConnection

In the case of a successful ModifyConnection message, the response line is followed by a LocalConnectionDescriptor, if the modification resulted in a modification of the session parameters (e.g., changing only the mode of a connection does not alter the session parameters). The LocalConnectionDescriptor is encoded as a “session description”, as defined in section 3.6.4. 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
```

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:

```
     00O 1207
```

3.6.3.3 DeleteConnection

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in Section 3.6.2.2.5

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

3.6.3.4 NotificationRequest

A NotificationRequest response does not include any additional response parameters.

3.6.3.5 Notify

A Notify response does not include any additional response parameters.

3.6.3.6 AuditEndpoint

In the case of an AuditEndpoint the response line may be followed by information for each of the parameters requested—each parameter will appear on a separate line. Parameters for which no value currently exists, e.g., digit map, will still be provided. Each local endpoint name “expanded” by a wildcard character will appear on a separate line using the “SpecificEndPointId” parameter code, e.g.:

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

or

```
200 1200 OK
A: a:PCMU;G728, p:10-100, e:on, s:off, t:1, v:L,
   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
```

3.6.3.7 AuditConnection

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

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

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

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

3.6.3.8 RestartInProgress

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

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

3.6.4 Session Description Encoding

The session description is encoded in conformance with the session description protocol ((SDP), refer to [4] for details of SDP), however, embedded clients may make certain simplifying assumptions about the session description as specified in the following. It should be noted, that session descriptions are case sensitive per RFC 2327.SDP usage depends on the type of session, as specified in the “media” parameter:

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

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

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

3.6.4.1 SDP Audio Service Use

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

1. The SDP profile presented below

2. RFC 2327 (SDP: Session Description Protocol)

The SDP profile provided describes the use of the session description protocol in NCS. The general description and explanation of the individual parameters can be found in RFC 2327, see [4], however below we detail what values NCS endpoints need to provide for these fields (send) and what NCS endpoints should do with values supplied or not supplied for these fields (receive).

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

Protocol Version (v=)

v= <version>
v= 0

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

Origin (o=)

The origin field consists (o=) of 6 sub-fields in RFC2327:

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

Username

Send: Hyphen MUST be used as username when privacy is requested.Hyphen SHOULD be used otherwise.[2]

Receive: This field SHOULD be ignored.

Session-ID

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

Receive: This field SHOULD be ignored.

Version

Send: In accordance with RFC 2327.

Receive: This field SHOULD be ignored.

Network Type

Send: Type 'IN' MUST be used.

Receive: This field SHOULD be ignored.

Address Type

Send: Type "IP4" MUST be used

Receive: This field SHOULD be ignored.

Address:

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

Receive: This field MUST be ignored.

Session Name (s=)

s= <session-name>

s= -

Send: Hyphen MUST be used as Session name.

Receive: This field MUST be ignored.

Session and Media Information (i=)

i= <session-description>

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

Receive: This field MUST be ignored.

URI (u=)

u= <URI>

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

Receive: This field MUST be ignored.

E-Mail Address and Phone Number (e=, p=)

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

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

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

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.

Bandwidth (b=)

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

Send: Bandwidth information is optional in SDP but it SHOULD always be included[3] . . When an rtpmap or a non well-known codec[4] is used, the bandwidth information MUST be used.
Receive: Bandwidth information SHOULD be included 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.

[3] If this field is not used, the Gate Controller might not authorize the appropriate bandwidth.
[4] A non well-known codec is a codec not defined in the PacketCable codec specification

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

Time, Repeat Times and Time Zones (t=, r=, z=)

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

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

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

Encryption Keys:

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

Security services for PacketCable are defined by the PacketCable Security specification [23]. The security services specified for RTP and RTCP do not comply with those of RFC 1889, RFC 1890, and RFC 2327. In the interest of interoperability with non-PacketCable devices, the “k” parameter will therefore not be used to convey security parameters.

Send: MUST NOT be used.

Receive: This field SHOULD be ignored.

Attributes (a=)

a= <attribute> : <value>
a= rtpmap : <payload type> <encoding name>/<clock rate> [/<encoding parameters>]
a= rtpmap : 0 PCMU / 8000
a= X-pc-codecs: <alternative 1> <alternative 2> …

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

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

rtpmap:

Send: When used, the field MUST be used in accordance with RFC 2327. It MAY be used for well-known as well as well as non well-known codecs. The encoding names used are provided in a separate PacketCable specification (see [21] and [23]).

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

X-pc-codecs:

Send: The field contains a list of alternative codecs that the endpoint is capable of using for this connection. The list is ordered by decreasing degree

of reference, 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 signaled through a media (m=) line.

Media Announcements (m=)

Media Announcements (m=) consists of 3 sub-fields:
M= <media> <port> <transport> <format>
M= audio 3456 RTP/AVP 0
Media:
Send: The ‘audio’ media type MUST be used.
Receive: The type received MUST be ‘audio’
Port:
Send: MUST be filled in accordance with RFC2327. The port specified is the receive port, regardless of whether the stream is unidirectional or bi-directional. The sending port may be different.
Receive: MUST be used in accordance with RFC 2327. The port specified is the receive port. The sending port may be different.
Transport:
Send: The transport protocol ‘RTP/AVP’ MUST be used.
Receive: The transport protocol MUST be ‘RTP/AVP’.
Media Formats:
Send: Appropriate media type as defined in RFC 2327 MUST be used.
Receive: In accordance with RFC 2327.

3.6.4.2 SDP Video Service Uses

Details on SDP use for video service have not been provided in the current version of NCS.

3.7 TRANSMISSION OVER UDP

3.7.1 Reliable Message Delivery

NCS messages are transmitted over UDP. Commands are sent to one of the IP addresses defined in the Domain Name System (DNS) for the specified endpoint or Call Agent. The responses are sent back to the source address of the command. However, it should be noted that the response may, in fact, come from another IP address than the one to which the command was sent. When no port is provisioned for the endpoint 31, the commands MUST be sent to the default NCS port, which is 2427 for Commands sent to Gateways and 2727 for commands sent to Call Agent. To minimize backward compatibility issues it is recommended that the Call Agent always explicitly state the NCS port to use in NCS messages (and not rely on the default). NCS messages, carried over UDP, may be subject to losses. In the

absence of a timely response, commands are repeated. NCS 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 Tthist 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 NCS entity does not execute the transaction, but simplyrepeats the response. If no match is found, the NCS entity examines the list of currently executing transactions. If a match is found, the NCS entity will not execute the transaction, which is simply ignored.

It is the responsibility of the requesting entity to provide suitable timeouts for all outstanding commands and to retry commands when timeouts have been exceeded. A retransmission strategy is specified below.Furthermore, when repeated commands fail to get a response, the destination entity is assumed to be unavailable.

3.7.2 Retransmission Strategy

This specification avoids specifying any static values for the retransmission timers since these values are typically network-dependent. Normally, the retransmission timers should estimate the timer by measuring the time spent between sending a command and the return of a response. Embedded clients MUST implement a transmission strategy using exponential back-off with configurable initial and maximum retransmission timer values. Embedded clients SHOULD use the algorithm implemented in TCP-IP, which uses two variables:

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

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

After any retransmission, the NCS 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.
  - RTOmax, where the default value for RTOmax 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.

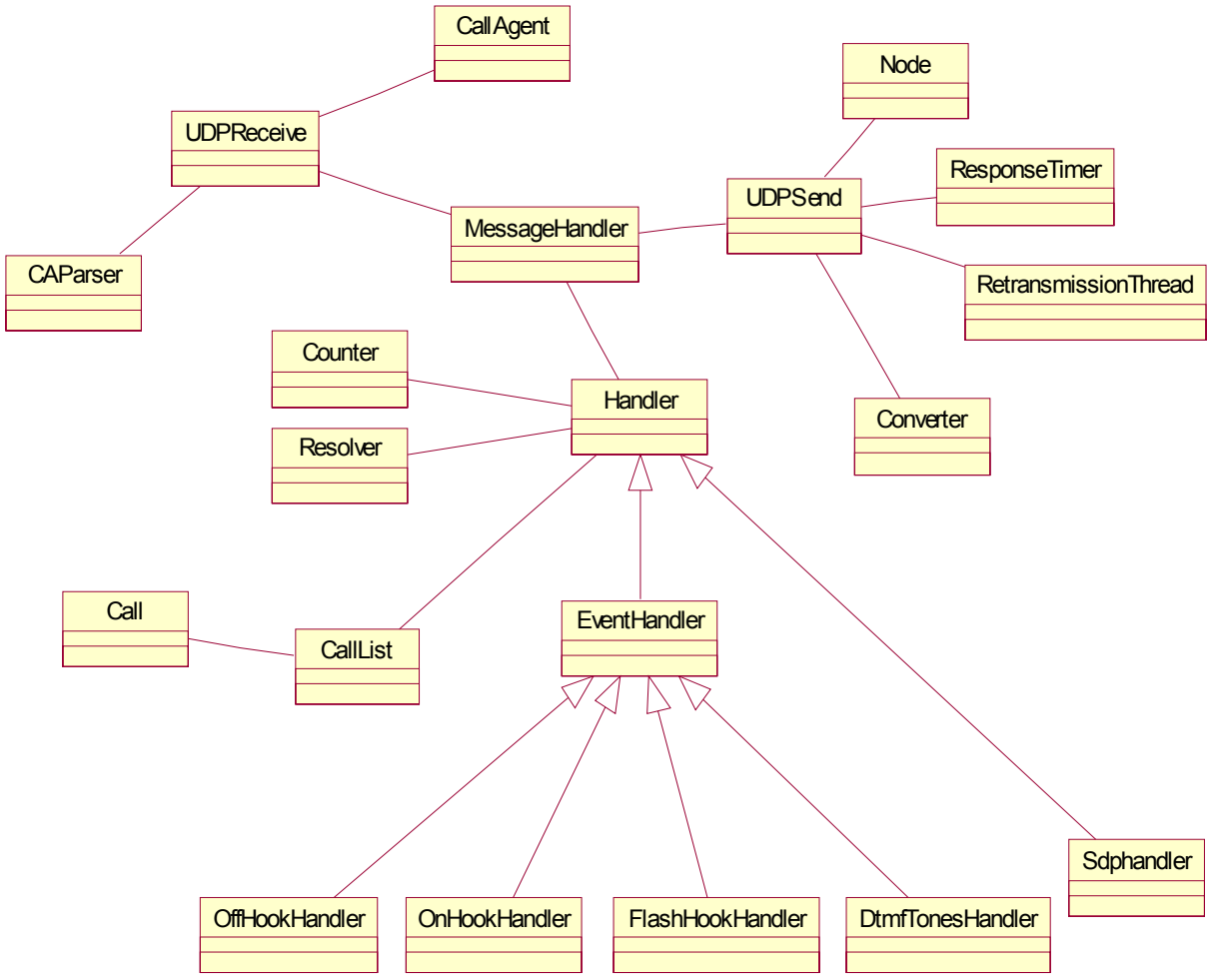
The initial value used for the retransmission timer is 200 milliseconds by default and the maximum value for the retransmission timer is 4 seconds by default. The provisioning process may alter these default values.

Chapter 4

Software Architecture

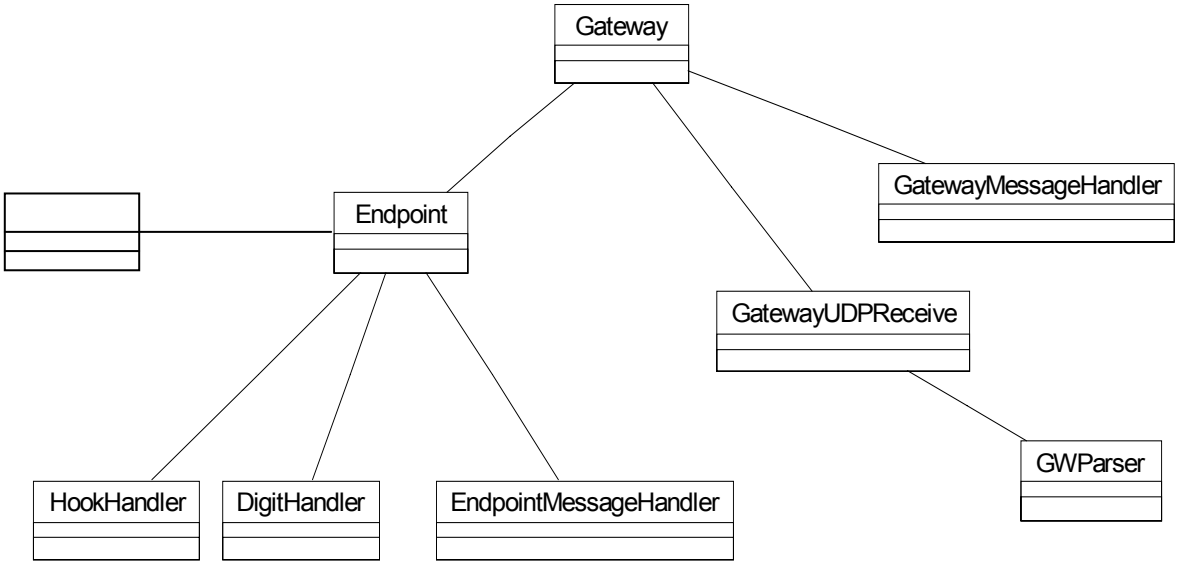
1 THE CALL AGENT

The following class design has been used to develop the Call agent ‘s functionality. Their details are in the next chapter.



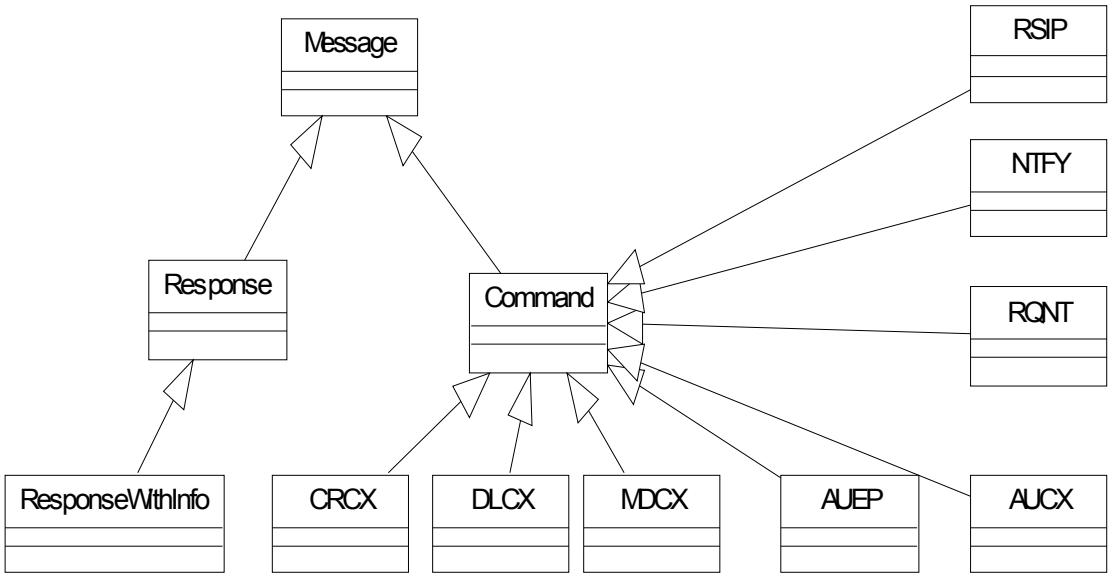
2 GATEWAY

The following classes implement the Gateways functionality.



SUPPORT CLASSES

The support classes support the Call agent and Gateway modules, they do not depict any functionality themselves



The following is a list of supporting classes which represent the parameters of the various NCS commands:

- RequestedEvent
- RequestedEvents
- EmbeddedRQNT
- EmbeddedMDCX
- SignalRequest
- SignalRequests
- ObservedEvent
- ObservedEvents
- DetectEvents
- Parameters
- LocalConnectionOptions
- Mode
- RequestedInfo
- SDP
- ConnectionParameters
- Capabilities
- Action

- Event
- Signal

Chapter 5

Software Implementation

INTERNAL FORMAT FOR COMMANDS

NCS is a text based protocol. Therefore all the commands being exchanged between the Call Agent and gateway are encoded in ASCII. However, processing these commands in this format is difficult, so an internal format has been established, in which each command is encapsulated by a class with the same name as the command. Whenever a command arrives at the Call Agent or at the gateway, it is translated into an object of its corresponding class. The same holds true for responses. The transformation from ASCII is achieved by the Parser (the CAParser in case of the Call Agent and the GWParser in case of the Gateway).

Similarly, commands are formulated inside the Call Agent or Gateway by instantiating objects of the corresponding classes. When these commands have to be transmitted over the network, they must first be translated into the appropriate ASCII strings. This is achieved by another module, called the Converter. After the commands have been transformed into ASCII, they can be transmitted as bytes, using UDP sockets.

Converter

The converter is responsible for converting the objects of different commands and responses into text before sending them over the UDP channels.

The UDPSend class calls the Converter class to convert the message(either command or response) object into string form. Inside the class it is first checked if the message is a command or a response , if command then what type of command is it . Since every command is different from the other and has its own parameters, so each command is treated differently for the conversion process in different methods. Following are the methods of the C*onverter* class.

ConvertMessage() :returns a string buffer,it is called before send() method is called in the send class.It further calls a series of overloaded methods.
Convert(NTFY nTFY):converts NTFY object into a text string.
Convert(RQNT rQNT):converts RQNT object into a text string.
Convert(AUCX aUCX):converts AUCX object into a text string.
Convert(AUEP aUEP):converts AUEP object into a text string.
Convert(DLCX dLCX):converts DLCX object into a text string.
Convert(CRCX cRCX):converts CRCX object into a text string.

Convert(MDCX mDCX):converts MDCX object into a text string.
Convert(Response response):converts Response object into a text string.
Convert(ResponseWithInfo responseWithInfo):converts ResponseWithInfo object into a string.

CheckAttExists

The Converter class uses this class for checking the existence of different parameters of commands or in other words checking if attributes of objects of different commands exist or not. This is checked because in every command, some parameters are optional and some are mandatory.

CheckAttExists class provides static overloaded methods for each of these to check whether they are assigned in the message object or not.
boolean checkAttExists(int[ ] IA):checks an integer array
boolean checkAttExists(String str): checks an string
boolean checkAttExists(String[ ] SA): checks an string array
boolean checkAttExists(ArrayList al): checks an array list

CallAgent

This class is responsible for generating the GUI of the call agent. description of the role played by all the classes involved and their interaction with each other is provided, as well as an overall view of the functioning of the Call Agent.

Message

The Message class represents all Messages that are transferred between the Call Agent and Gateway. This includes both Commands and their Responses.

Command

The Command class represents all Commands and is derived from the Message class. The eight classes that are derived from Command represent the eight commands of NCS as shown below:

CRCX represents CreateConnection command.
MDCX represents ModifyConnection command.
DLCX represents DeleteConnection command.
RQNT represents NotificationRequest command.
NTFY represents Notify command.
RSIP represents RestartInProgress command.
AUEP represents AuditEndpoint command.
AUCX represents AuditConnection command.

Response

The Response class represents Responses and is also derived from the Message class. There are two kinds of responses which are possible, a response which carries no information with it and a response which contains additional information. The original Response class encapsulates the simple response without any information, which just contains a ReturnCode and a TransactionIdentifier.

ResponseWithInfo

This class is derived from Response and encapsulates all responses which carry additional information.

SUPPORTING CLASSES

In NCS each command contains a number of parameters. These parameters accompany the command and can have different values depending on how the Call Agent wants the gateway to behave. Some parameters can have a list of values associated with them. Therefore, wherever required, classes have been created to encapsulate these parameters.
The following is a list of supporting classes which represent the parameters of the various commands:

RequestedEvent

This class specifies a single requested event within the list of requested events which the CA sends to the gateway in a NotificationRequest. It encapsulates the event, the action associated with the event and other parameters which may be associated with the event.

RequestedEvents

The list of requested events sent by the Call Agent in a NotificationRequest.

EmbeddedRQNT

Encapsulates the EmbeddedRQNT action that can be specified with an event.

EmbeddedMDCX

Encapsulates the EmbeddedRQNT action that can be specified with an event.

SignalRequest

Encapsulates a single signal, which is to be applied to an endpoint. It can also specify the time-out period for the signal as well as other parameters associated with the signal.

SignalRequests

The list of signal requests sent by the Call Agent in a NotificationRequest.

ObservedEvent

A single event which the Gateway has observed and its associated parameters.

ObservedEvents

List of observed events present in a Notification.

DetectEvents

Minimum list of events which the gateway must detect when it is in certain states.

Parameters

Encasulates the list of parameter codes for the various command parameters.

LocalConnectionOptions

Describes the characteristics of the media connection such as encoding method, packetization period etc.

Mode

Used to associate a mode with a connectionId.

RequestedInfo

The list of parameters requested by the Call Agent when it audits an Endpoint.

SDP

Encapsulates the session description in conformance with the Session Description Protocol. The SDP contains information such as the addresses and RTP ports which are necessary for establishing a media connection.

ConnectionParameters

Encapsulates the performance parameters associated with a connection such as the packets sent, packets received etc.

Capabilities

Encapsulates the capabilities of the endpoint such as the packages and modes supported.

Action

Encapsulates all the actions which can accompany a ReqeustedEvent.

Event

Encapsulates all the events in the Line Package.

Signal

Encapsulates all the Signals in the Line Package.

INTERNAL WORKING OF THE CALL AGENT

CallAgent

This is the class containing the main method, where the processing begins. It instantiates an object of the UDPReceive class and invokes its receiveMessage() method so that the Call Agent can begin to listen for messages from the Gateways.
This class is also responsible for creating the GUI of the Call Agent. The Call Agent is provided with a GUI so that its internal working, and the state of all the calls that are being managed by it at a point in time can be examined by the user.

UDPReceive

This class is responsible for listening for the arrival of messages from the Gateways. The messages which arrive are in the form of ASCII encoded strings (since NCS is a text based protocol). Messages refer to both NCS commands and their responses. The UDPReceive class after receiving a message, hands it over to the CAParser (CallAgent Parser) which translates the message into an object of its corresponding class.After this, the UDPReceive class passes the message to the MessageHandler so that the Call Agent can begin to process the message.

CAParser

This class translates the ASCII received NCS messages by the Call Agent into objects of appropriate classes of NCS commands.
The parser uses a lexical analyzer, which tokenizes the recieved message, and on the basis of these tokens the validity of message according to the protocol syntax is checked.The main and important methods of CAParser are:

parse( ) : called by the UDPRecieve class when it receives a message.

parseRSIP( ) : parses the RestartInProgress command
parseNTFY( ) : parses the Notify command
parseDLCX( ) : parses the DeleteConnection command
parseResponse( ) : parses the Response

lexan( ) ,mgcpCommand( ) , mgcpResponse( ) , mgcpCommandLine( ), mgcpResponseLine( ), mgcpParameter, parseSdp( ) etc. are also some important methods.

MessageHandler

On receiving a message the Receive class passes it on to the MessageHandler which then determines what kind of message it is and how it should be processed. The MessageHandler has the following overloaded methods:

- Handle(Message)
  The Receive class calls this method to pass the Message object returned by the CAParser to the MessageHandler. This method determines what king of object it is (i.e NTFY, RSIP etc) and passes it to the appropriate overloaded method.

- Handle(NTFY)
  This method determines what ObservedEvents the Notification object contains and calls the appropriate Handler (i.e. OffHookHandler if the event reported is OffHook).

- Handle(RSIP)
  This method carries out the Registration Process.

- Handle(Response)
  This method is responsible for handling Resposes.

- Handle(ResponseWithInfo)
  When a ResponseWithInfo arrives and it contains a SDP, the SdpHandler is called by this method.

Handler

This class and its child classes encapsulate the core functionality of the call agent- which means responding to events occurring at endpoints and sending appropriate commands in response to these events.

EventHandler extends Handler
Its child classes handle all the incoming events which are reported to the CallAgent by the Gateway.

Call

The Call Agent needs to maintain information about the Calls which are being managed by it. It needs to track the progress of each call. This class enapsulates all the information which is required to moniter a single call. This includes the following:

- **endpoint1**: the endpointId of the caller
- **connectionId1**: the connectionId of the caller
- **mode1**: the connection mode of the caller
- **s1**: the Session description for the caller.
- **endpoint2**: the endpointId of the callee
- **connectionId2**: the connectionId of the callee
- **mode2**: the connection mode of the callee
- **s2**: the Session Description for the callee
- **callId**: the callId of the entire call
- **boolean callInProgress**: whether the the call has started or not (i.e both ends of the connection are in sendreceive mode)

CallList

This is the data structure which contains the list of calls which are currently going on.

OffHookHandler

This class is responsible for handling the off hook event, i.e when the user picks up the receiver at an endpoint. When such an event takes place, the endpoint will form and send a Notification command to the Call Agent. The OffHookHandler will eventually be invoked inside the Call Agent when it receives this Notification. The OffHookHandler will search through the list of calls present in the CallList.
If the endpoint is not present in the CallList, a new connection has to be established at that endpoint. Therefore a CRCX is sent to it.
If the endpoint is present and it is the callee, an MDCX is sent to the corresponding caller (to change the connection mode to sendreceive and stop ringbacktones). Also an RQNT is sent to the endpoint itself so that it stops ringing. Refer to the Normal Call Flow Appendix D.

DtmfTonesHandler

- This class is responsible for handling any digits which the user dials. It checks the CallList to see if the endpoint is the caller. It then sends an RQNT to the endpoint to stop digit collection
- It maps the digits to the corresponding endpointId by calling the Resolver method getIPAddress().

- If the resulting endpointId is invalid, an error message is displayed .
- If the endpointId is the callers own then an RQNT to apply busy tones is sent to the caller.
- If the resulting endpoint is already conversing with two other endpoints then an RQNT to apply busy tones is sent to the caller.
- If the resulting endpoint is only conversing with one other endpoint then a CRCX is sent to it along with RQNT to apply callwaiting tones.

Finally, if the resulting endpointId is valid and belongs to an endpoint which is not involved in any call, then a CRCX is sent to it, along with RQNT to apply ringing.

SdpHandler

On the arrival of a Session Description (SDP) this class must trigger certain events. When a Session Description arrives from the callee, it has to relay it to the caller. Ringback tones also have to be applied at the caller. Therefore, it sends an MDCX along with RQNT to the caller.

OnHookHandler

This class handles the on hook event, i.e when the user puts the phone back on the hook. It sends commands for deleting the connections which were present at the endpoints.
The DLCX command is sent to both the endpoints involved in the connection. It is sent for all connections which were present at the endpoint.

FlashHookHandler

This class handles the flash hook event, i.e when the user briefly presses the phone hook.
This event is associated with the Call Waiting Call Flow. MDCX commands are sent in order to activate or inactivate connections at the endpoints. The currently active connection is made inactive, whereas the inactive connection is made active.

Counter

The counter class provides cyclical counters for the following:

- TransactionId
- CallId
- ConnectionId
- RequestIdentifier

Resolver

This class maps the phone numbers dialed by the user to the endpointIds.

UDPSend

Responsible for sending messages from the CA (CallAgent) to the gateway and vice versa. It is also responsible for maintaining retransmission timers and retransmitting messages if a timeout occurs.

GATEWAY

Gateway

This is the main class in the case of the gateway, which contains the main method and where the processing for the gateway begins. It instantiates all the endpoints which will be present at the gateway. Currently, four endpoints are associated with a gateway, for testing puposes.

The gateway has mainly been implemented in order to test and demonstrate the working of the Call Agent. Since the basic purpose of the Call Agent is to implement the basic Call Flows (i.e. the sequence of commands responsible for enabling an entire call to take place), and these Call Flows involve commands from both the Call Agent, and the gateway, a rudimentary implementation of the gateway is necessary in order to show the operation of the Call Agent.
A proper and complete implementation of the gateway is hardware related and outside the scope of the project.
As part of its working, the Gateway class instantiates an object of the atewayUDPReceive class so that the process of receiving commands from the Call Agent can begin.

GatewayUDPReceive

This class is responsible for listening for messages from the Call Agent. After receiving a command, it routes it to the appropriate endpoint according to the endpointId parameter contained in the command. If the endpointId parameter is wildcarded, however, it is passed to the GatewayMessageHandler class.

GWParser

This module represents the parser for the gateway side. It validates all the commands which the gateway receives and converts them into objects of NCS commands.

The GWparser uses a lexical analyzer (different from that of CAParser) which tokenizes the recieved message and on the basis of these tokens the validity of message according to the protocol syntax is checked.
The following is how it achieves this :

The type of received message is checked by the first token in the parse method I.e. is it a Create Connection command, Delete Connection command etc. or a response, then the appropriate method is invoked on the basis of command.
The main and important methods of CAParser are:

parse( ) : called by the UDPRecieve class when it receives a message.
parseCRCX( ) : parses the RestartInProgress command
parseMDCX( ) : parses the Notify command
parseDLCX( ) : parses the DeleteConnection command
parseRQNT( ) : parses the NotificationRequest command
..
..
..
parseResponse( ) : parses the Response

lexan( ), match( ) ,mgcpCommand( ) , mgcpResponse( ) , mgcpCommandLine( ), mgcpResponseLine( ), mgcpParameter, parseSdp( ) etc. are also some important methods.

GatewayMessageHandler

When the Call Agent sends a command in which the endpointId has been wildcarded, the GatewayMessageHandler is responsible for processing it.

Endpoint

This class encapsulates the working of an endpoint. It is responsible for the GUI of the endpoint.

HookHandler

This class is responsible for detecting the occurrence of events like on hook, off hook and flash hook, at an endpoint. When one of these events takes place, it formulates a Notification and sends it to the Call Agent. Since all these events are persistent, the HookHandler will continuously look for the presence of these events at the endpoint.

DigitHandler

This class is responsible for collection of the digits dialed by the user. This is not a persistent event and it is detected only when it is requested by the Call Agent in a NotificationRequest.

EndpointMessageHandler

After an endpoint receives a message from the gateway it is passed to this class in order to be processed.

DigitMap

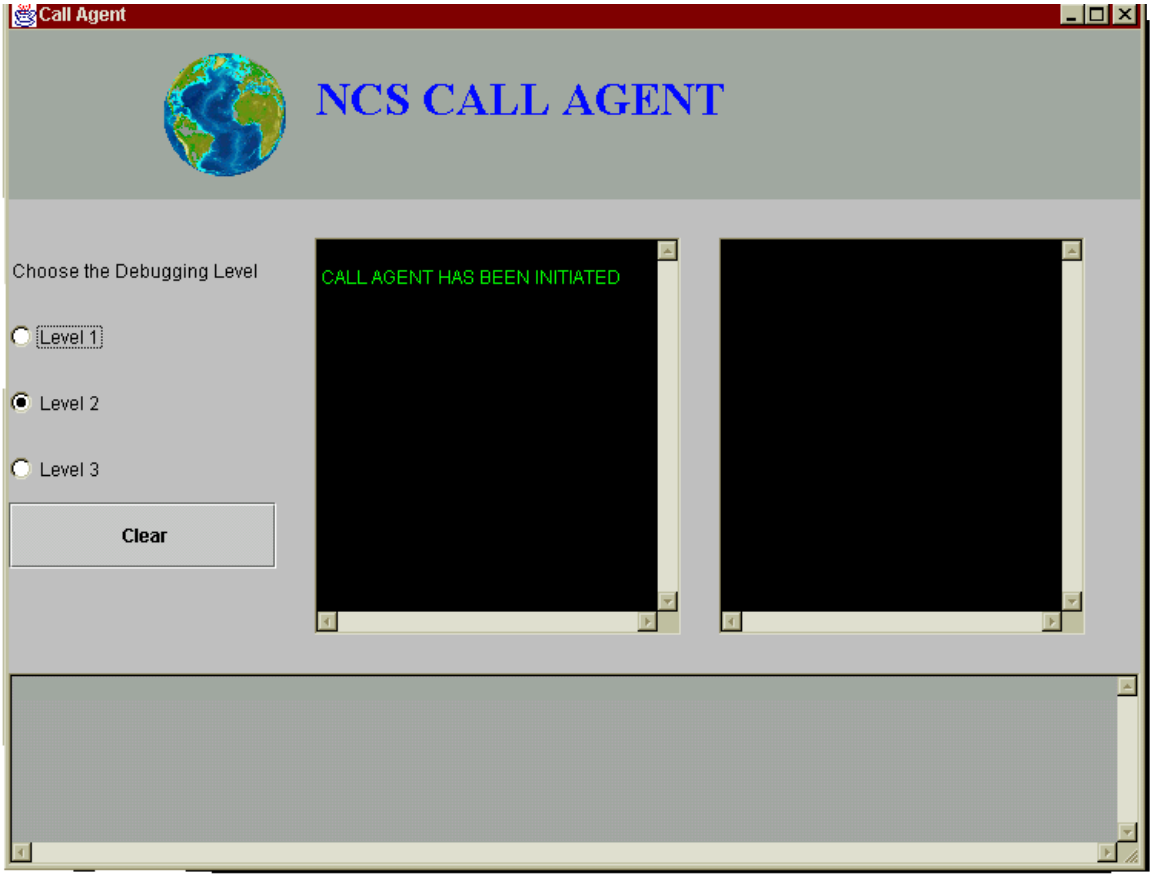
The DigitMap class matches the digits accumulated by the endpoint with the DigitMap provided by the CallAgent.

Chapter 6

User Manual

For using and testing the software, a Java-code running environment is required with jdk 1.3 version of the Java Development Toolkit.

With the Call Agent on one PC and the Gateway software on two different PCs, we first run the Call Agent and get the following user interface. The left text area displays the text messages received the call Agent. The received messages can be viewed with other details depending on the level of debugging chosen. The level 1 is an abstract level description, while the next levels get more detailed and technical. Text areas can be cleared with the clear button. The right text area displays the retransmitted messages. The text area at the lower portion displays, which endpoints are busy in a call at a time.



Running the gateway software generates 4 of the following user devices(telephone sets) simulation. By clicking on the ‘On hook’ button, the user gets ‘off hook’ , he then hears a dial tone and can dial a number. The text area in the upper left corner displays the signals applied to the phone like dial tone, ringing, call waiting tones etc. The ‘flash ‘ button is used when the user gets call waiting tones while in call with a second user. The tones indicate that a third person wants to talk to him. He can switch to this person by using flash button, thereby inactivating his current call and switching to another one. To switch back to the original call, flash button is again used.

Summary

ACHIEVED RESULTS AND CONCLUSIONS

We started the project with the objectives mentioned in the project statement and achieved all of them. We have implemented a Call Agent that is controlling and monitoring two media gateways, each supporting 4 endpoints, but it is capable of controlling n number of gateways supporting n number of endpoints.

The Call Agent has been tested with simulated gateways (which are not fully implemented ones). Simulated gateways were developed since our objective was not the implementation of gateway but the Call Agent. The gateways can notify the occurrence of different events to the Call Agent and the Call Agent can then take corresponding (required) steps.

The project allowed us to implement our concepts and knowledge gained during almost all of our undergraduate degree courses. For example, our Software Engineering course helped us in planning and organizing our background study, requirement analysis, design, coding and testing phases. Compilers course was practically used in developing the parser. The Data Structures, Java, Technical Writing, Computer Networks and other courses also helped us a lot.

This project was a very good experience and has helped us in gaining valuable knowledge about the different protocol standards, their use, Networking issues and the growing market of Voice over IP market.

FUTURE RECOMMENDATIONS

We would like to recommend the following features to be incorporated in our developed software if any one else wants to continue with its further development in future:

- Three way call
- Conference call
- Billing features of the Call Agent
- Security features of the NCS Protocol (see [5])

Appendix A

Event Packages

This section defines a set of event packages for the various types of endpoints currently defined by PacketCable for embedded clients. Each package defines a package name for the package and event codes and definitions for each of the events in the package. In the tables of events/signals for each package, there are five columns:

Code	The package unique event code used for the event/signal.
Description	A short description of the event/signal.
Event	A check mark appears in this column if the event can be requested by the Media Gateway Controller. Alternatively, one or more of the following symbols may appear:
"P"	indicating that the event is persistent,
"S"	indicating that the event is an event-state that may be audited,
"C"	indicating that the event/signal may be detected/applied on a connection.
Signal	If nothing appears in this column for an event, then the event cannot be signaled on command by the Media Gateway Controller. Otherwise, the following symbols identify the type of event:
"OO"	On/Off signal. The signal is turned on until commanded by the Media Gateway Controller to turn it off, and vice versa.
"TO"	Timeout 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.

Analog Access Lines

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

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

Code	Description	Event	Signal	Additional Info
0-9,*,#,A,B,C,D	DTMF tones	✔	BR	
bz	Busy tone	-	TO	Time-out = 30 seconds
cf	Confirmation tone	-	BR	
ci(ti, nu, na)	Caller Id	-	BR	“ti” denotes time, “nu” denotes number, and “na” denotes name
dl	Dial tone	-	TO	Time-out = 16 seconds
ft	Fax tone	✔	-	
hd	Off-hook transition	P,S	-	
hf	Flash hook	P	-	
hu	On-hook transition	P,S	-	
L	DTMF long duration	✔	-	
ld	Long duration connection	C	-	
ma	Media start	C	-	
mt	Modem tones	✔	-	
mwi	Message waiting Indicator	-	TO	Time-out = 16 seconds
oc	Operation complete	✔	-	
of	Operation failure	✔	-	
ot	Off-hook warning tone	-	TO	Time-out = infinite
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive ringing (0..7)	-	TO	Time-out = 180 seconds
rg	Ringing	-	TO	Time-out = 180 seconds
ro	Reorder tone	-	TO	Time-out = 30 seconds
rs	Ringsplash	-	BR	
rt	Ring back tone	-	C,TO	Time-out = 180 seconds
sl	Stutter dial tone	-	TO	Time-out = 16 seconds
t	Timer	✔	-	

TDD	Telecomm Devices for the Deaf (TDD) tones	✔	-	
vmwi	Visual message waiting indicator	-	OO	
wt1, wt2, wt3, wt4	Call waiting tones	-	TO	Time-out = 12 seconds
X	DTMF tones wildcard	✔	-	Matches any of the digits “0-9”

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

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

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

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

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

- The **time** parameter is coded as “MM/DD/HH/MM”, where MM is a two digit value for Month between 01 and 12, DD is a two-digit value for Day between 1 and 31, and Hour and Minute are two-digit values coded according to military local time, e.g., 00 is midnight, 01 is 1 a.m., and 13 is 1 p.m.
- The **number** parameter is coded as an ASCII character string of decimal digits that identify the calling line number. White spaces are permitted if the string is quoted, however they will be ignored.

- The **name** parameter is coded as a string of ASCII characters that identify the calling line name. White spaces are permitted if the string is quoted.

A "P" in the number or name field is used to indicate a private number or name, and an "O" is used to indicate an unavailable number or name. The following example illustrates the use of the caller-id signal:

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

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

Fax tone (ft): The fax tone event is generated whenever a fax call is detected by presence of V.21 fax preamble. The fax tone event SHOULD also be generated when the T.30 CNG tone is detected. See ITU-T Recommendations T.30 and V.21.

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

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

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

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

Long duration connection (ld): The "long duration connection" is detected when a connection has been established for more than a certain period of time. The default value is 1 hour, however this may be changed by the provisioning process.
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[5] 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.

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

Modem tones (mt): Modem tone (mt): The modem tone event is generated whenever a data call is detected by presence of V.25 answer tone (ANS) with or without phase reversal or V.8 modified answer tone (ANSam) with or without phase reversal. See ITU-T Recommendation V.25 and V.8.

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

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

```
O: L/oc(L/dl)
```

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

```
O: L/oc(L/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[6]:

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

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

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

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

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

When the operation failure event is requested, event parameters can not be specified.When the package name is omitted, the default package name is assumed. The operation failure event may additionally be generated as specified in the base protocol, e.g. when an embedded ModifyConnection command fails, as in

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

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

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

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

Ringing (rg): See GR-506-CORE – LSSGR: SIGNALING, Section 14. The provisioning process may define the ringing cadence. The ringing signal may be parameterized with the signal parameter "rep" which specifies the maximum number of ringing cycles (repetitions) to apply. The following will apply the ringing signal for up to 6 ringing cycles:

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

It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

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

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

Ring back tone (rt): Audible Ring Tone is a combination of two AC tones with frequencies of 440 and 480 Hertz and levels of –19 dBm each, to give a combined level of –16 dBm. In the U.S. the cadence for Audible Ring Tone is defined to be 2 seconds on followed by 4 seconds off. The definition of the tone is defined by the national characteristics of the Ringback Tone, and MAY be established via

provisioning. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.5.
The ringback signal can be applied to both an endpoint and a connection.
When the ringback signal is applied to an endpoint, it is considered an error to try and play ring back tones, if the endpoint is considered on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook). When the ringback signal is applied to a connection, no such check is to be made.

Stutter Dial tone (sl): Stutter Dial Tone (also called Recall Dial Tone) is generated by supplying Confirmation Tone, followed by continuous Dial Tone. See GR-506-CORE – LSSGR: SIGNALING, Section 17.2.2. The stutter dial tone signal may be parameterized with the signal parameter "del" which will specify a delay in milliseconds to apply between the confirmation tone and the dial tone[7] . The following will apply stutter dial tone with a delay of 1.5 seconds between the confirmation tone and the dial tone:

```
S: sl(del=1500)
```

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

Timer (t): As described in Section 4.1.4, timer T is a provisionable timer that can only be cancelled by DTMF input. When timer T is used with the "accumulate according to digit map" action, the timer is not started until the first digit is entered, and the timer is restarted after each new digit is entered until either a digit map match or mismatch occurs. In this case, timer T functions as an inter-digit timer and takes on one of two values, Tpar or Tcrit. When at least one more digit is required for the digit string to match any of the patterns in the digit map, timer T takes on the value Tpar, corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value Tcrit corresponding to critical timing. An example
use is:

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

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

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

Note, that only one of the two forms can be used at a time, since a given event can only be specified once.
The default value for Tpar is 16 seconds and the default value for Tcrit is 4 seconds. The provisioning process may alter both of these.

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

[7] This feature is needed for, e.g., Speed Dialing.

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

Call Waiting tone1 (wt1, .., wt4): Call Waiting tones are defined in GR-506-CORE, Section 14.2 – the number refers to the tone pattern used. GR-571-CORE (FSD 01-02-1201) indicates that two tone patterns should be played separated by a period of 10 seconds. The default Call Waiting tone is a 440-Hz tone applied for 300 ± 50 ms. The talking path should be interrupted for a maximum of 400 ms for the application of each CW tone pattern. When this signal is requested, the embedded client will playthe two tone patterns as specified in GR-571-CORE before the "TO" signal times out. It is considered an error to try and apply call waiting tones on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

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

Video

Event packages for video have not been provided in the current version of NCS document.

Appendix B

Return Codes and Error Codes

Return Codes and Error Codes

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

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

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

Code	Meaning
000	Response acknowledgement.
100	The transaction is currently being executed. An actual completion message will follow later.
200	The requested transaction was executed normally.
250	The connection(s) was deleted.
400	The transaction could not be executed, due to a transient error.
401	The phone is already off hook.
402	The phone is already on hook.
500	The transaction could not be executed because the endpoint is unknown.
501	The transaction could not be executed because the endpoint is not ready.
502	The transaction could not be executed because the endpoint does not have sufficient resources.
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
510	The transaction could not be executed because a protocol error was detected.

511	The transaction could not be executed because the command contained an unrecognized extension.
512	The transaction could not be executed because the gateway is not equipped to detect one of the requested events.
513	The transaction could not be executed because the gateway is not equipped to generate one of the requested signals.
514	The transaction could not be executed because the gateway cannot send the specified Announcement.
515	The transaction refers to an incorrect connection-id (may have been already deleted).
516	The transaction refers to an unknown call-id.
517	Unsupported or invalid mode.
518	Unsupported or unknown package.
519	Endpoint does not have a digit map.
520	The transaction could not be executed because the endpoint is “restarting”.
521	Endpoint redirected to another Call Agent.
522	No such event or signal
523	Unknown action or illegal combination of actions.
524	Internal inconsistency in LocalConnectionOptions
525	Unknown extension in LocalConnectionOptions
526	Insufficient bandwidth
527	Missing RemoteConnectionDescriptor
528	Incompatible protocol version
529	Internal hardware failure
532	Unsupported value(s) in LocalConnectionOptions
533	Response too big

Appendix C

Reason Codes

Reason Codes

Reason-codes are used by the gateway when deleting a connection to inform the Call Agent about the reason for deleting the connection. The reason code is an integer number, and the following values have been defined:

Code	Meaning
900	Endpoint malfunctioning
901	Endpoint taken out of service
902	Loss of lower layer connectivity (e.g., downstream sync)
903	QoS resource reservation was lost.

Appendix D

EXAMPLE CALL FLOW

The following is a **Normal call flow** between two embedded clients EC-1 and EC-2 and the Call agent is represented by CA. CDB is the Configuration Database which we have implemented through a Resolver class.

User-1	EC-1	CA	CDB	EC-2	User-2
	←	**RQNT**			
	Ack---------	→			
offhook	**Notify**	→			
	←	**Ack**			
dialtone	←	**CRCX+RQNT**			
	Ack SDP1	→			
digits	**Notify**	→			
	←	**Ack**			
prog-ress	←	**RQNT**			
	Ack--------	→			
		Query(E.164)-	→		
		←	**IP**		
		CRCX(SDP1)+RQNT--------	**------**	→	
		←	**------**	**Ack SDP2**	**ringing**

User-1	EC-1	CA	CDB	EC-2	User-2
ring-back	←	**MDCX(SDP2)+ RQNT**			
	Ack--------	→			
		←	------	**Notify**	**offhook**
		Ack----------	------	→	
	←	**MDCX+RQNT**			
	Ack--------	→			
		Call Start			
		RQNT----------	------	→	
		←	------	**Ack**	
		Call Established			
		←	------	**Notify**	**onhook**
		Ack----------	------	→	
		DLCX----------	------	→	
	←	**DLCX**			
	Ack--------	→	←	**Ack**	
		RQNT----------	------	→	
			←	**Ack**	
onhook	**Notify-----**	→			
	←	**Ack**			

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

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

```
RQNT 1201 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AB
R: hd
```

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

```
200 1201 OK
```

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

```
NTFY 2001 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AB
O: hd
```

The Call Agent immediately acknowledges the notification:

```
200 2001 OK
```

The Call Agent examines the services associated to an off hook event for this endpoint (it could take special actions in the case of a direct line, no current subscription, etc.). In most cases, it will send a combined CreateConnection and NotificationRequest command to create a connection, provide dial-tone, and collect DTMF digits [8] :

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

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

```
200 1202 OK
```

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

```
I: FDE234C8
v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-c=
IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

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

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

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

The Call Agent immediately acknowledges that notification.

```
200 2002 OK
```

At this stage, the Call Agent will send a NotificationRequest, to stop collecting digits yet continue to watch for an on-hook transition. The Call Agent furthermore decides to acknowledge receipt of the responses for transaction 1202:

```
RQNT 1203 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
K: 1202
X: 0123456789AD
R: hu
```

The embedded client immediately acknowledges that command.

```
200 1203 OK
```

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

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

```
S: rg


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

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

The endpoint identifier points towards the outgoing circuit.
The message carries the session description returned by the ingress embedded client,
Because the session description is present, the “mode” parameter is set to “send/receive”.

We observe that the call identifier is identical for the two connections. This is normal since the two connections belong to the same call.
We assume, this command does not finish executing immediately [9] , and a provisional
response is therefore returned by the egress embedded client acknowledging the command, sending in the session description its own parameters such as address, ports and RTP profile as well as the connection identifier for the new connection:

```
100 2001 Pending
I: 32F345E2


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

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

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

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

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

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

```
000 2001
```

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

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


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

The embedded client immediately acknowledges the modification:

```
200 1204 OK
```

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

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

The call agent immediately acknowledges that notification.

```
200 3001 OK
```

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

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

The embedded client immediately responds to the command:

```
200 1206 OK
```

In parallel, the Call Agent asks the egress embedded client to notify the occurrence of an on-hook event. It does so by sending a NotificationRequest to the embedded client [10] :

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

The embedded client immediately responds to the command:

```
200 2002 OK
```

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

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

The Call Agent immediately responds to the command:

```
200 2003 OK
```

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

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

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

```
I: FDE234C8

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

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

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

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

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

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

The embedded client will acknowledge this message:

```
200 2005 OK
```

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

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

The Call Agent immediately responds to the command:

```
200 1208 OK
```

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

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

The embedded client will acknowledge this message:

```
200 1209 OK
```

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

The **Call waiting call flow** is the same as the Normal call flow except for a third party, User-3 on EC-3, who tries to call on User-1 in call with User-2, with the result ,the User-1 gets Call waiting tones , on flash-hook ,User-1 can switch to User-3 while User-2 connection remains inactive for that time period. When User-1 does flash-hook again, he can switch back to his original call with User-2 while the third party User-3 goes out of scene.

Appendix E

FORMAL SYNTAX DESCRIPTION

NETWORK-BASED CALL SIGNALING (NCS) PROTOCOL'S GRAMMAR

In this section, we provided a formal description of the protocol syntax (reference [2]), following the "Augmented BNF for Syntax Specifications" defined in RFC 2234.

```
MGCPMessage = MGCPCommand / MGCPResponse

MGCPCommand = MGCPCommandLine 0*(MGCPParameter)
              [EOL *SDPinformation]

MGCPCommandLine = MGCPVerb 1*(WSP) <transaction-id> 1*(WSP)
                  <endpointName> 1*(WSP) MGCPversion EOL

MGCPVerb = "CRCX" / "MDCX" / "DLCX" / "RQNT" /"NTFY" /
           "AUEP" / "AUCX" / "RSIP" / extensionVerb

extensionVerb = "X" 3(ALPHA / DIGIT)

transaction-id = 1*9(DIGIT)

endpointName =  localEndpointName "@" DomainName
localEndpointName = localNamePart 0*("/" LocalNamePart)
localNamePart = AnyName / AllName / NameString
AnyName = "$"
AllNames = "*"
NameString = 1*(range-of-allowed-characters)
DomainName = 1*256(ALPHA / DIGIT / "." / "-"); as defined
             in RFC 821

MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
              [1*(WSP) ProfileName]
ProfileName = "NCS" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)

MGCPParameter = ParameterValue EOL

ParameterValue = ("K" ":" 0*WSP <ResponseAck>) /
                 ("C" ":" 0*WSP <CallId>) /
```

```
                    ("I" ":" 0*WSP <ConnectionId>) /
                    ("N" ":" 0*WSP <NotifiedEntity>) /
                    ("X" ":" 0*WSP <RequestIdentifier>) /
                    ("L" ":" 0*WSP <LocalConnectionOptions>) /
                    ("M" ":" 0*WSP <ConnectionMode>) /
                    ("R" ":" 0*WSP <RequestedEvents>) /
                    ("S" ":" 0*WSP <SignalRequests>) /
                    ("D" ":" 0*WSP <DigitMap>) /
                    ("O" ":" 0*WSP <ObservedEvents>) /
                    ("P" ":" 0*WSP <ConnectionParameters>) /
                    ("E" ":" 0*WSP <ReasonCode>) /
                    ("Z" ":" 0*WSP <SpecificEndpointId>) /
                    ("ZN" ":" 0*WSP <NumEndpoints>) /
                    ("ZM" ":" 0*WSP <MaxEndpointIds>) /
                    ("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>) /
                    (extensionParameter ":" 0*WSP
                    <parameterString>)

ResponseAck =  confirmedTransactionIdRange
               *["," confirmedTransactionIdRange ]

confirmedTransactionIdRange = 1*9DIGIT [ "-" 1*9DIGIT ]

CallId = 1*32(HEXDIG)

//  The  audit  request  response  may  include  a  list  of
identifiers
ConnectionId = 1*32(HEXDIG) 0*("," 1*32(HEXDIG))

NotifiedEntity = [LocalName "@"] DomainName [":"portNumber]
LocalName = 1*32(suitableCharacter)
portNumber = 1*5(DIGIT)

RequestIdentifier = 1*32(HEXDIG)

LocalConnectionOptions = [LocalOptionValue 0*(WSP)
                  0*("," 0*(WSP) LocalOptionValue 0*(WSP)) ]
LocalOptionValue = ("p" ":" <packetizationPeriod> )
                  / ("a" ":" <compressionAlgorithm> )
                  / ("e" ":" <echoCancellation> )
                  / ("s" ":" <silenceSuppression> )
                  / ("t" ":" <typeOfService> )
```

```
                / (LocalOptionExtensionName ":"
                / localOptionExtensionValue)

Capabilities = [CapabilityValue 0*(WSP)
               0*("," 0*(WSP) CapabilityValue 0*(WSP))]


CapabilityValue = LocalOptionValue
               / ("b" ":" <bandwidth> )

                / ("v" ":" <supportedPackages>)
                / ("m" ":" <supportedModes> )

packetizationPeriod = 1*4(DIGIT)["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
algorithmName = 1*32(SuitableCharacter)
bandwidth = 1*4(DIGIT)["-" 1*4(DIGIT)]
echoCancellation = "on" / "off"
silenceSuppression = "on" / "off"
typeOfService = 2HEXDIG
supportedModes= ConnectionMode 0*(";" ConnectionMode)
supportedPackages = packageName 0*(";" packageName)

localOptionExtensionName = "x"("+"/"-")
                           1*32(SuitableCharacter)

localOptionExtensionValue = 1*32(SuitableCharacter) /
                            quotedString

ConnectionMode = "sendonly" / "recvonly" / "sendrecv" /
                 "confrnce" / "inactive" / "replcate" /
                 "netwloop" / "netwtest"

RequestedEvents = [requestedEvent 0*("," 0*(WSP)
                   requestedEvent)]
requestedEvent = eventName [ "(" requestedActions ")" ]

eventName = [ (packageName / "*") "/" ] (eventId / "all" /
            eventRange)[ "@" (ConnectionId / "$" / "*") ]
packageName = "L" / "S"
eventId = 1*(SuitableCharacter)
eventRange = "[" 1*(DIGIT / DTMFLetter / "*" / "#" /
             (DIGIT "-" DIGIT)/(DTMFLetter "-" DTMFLetter))
             "]"

requestedActions = requestedAction 0*("," 0*(WSP)
                   requestedAction)
```

```
requestedAction = "N" / "A" / "D" / "I" / "K" /
                  "E" "(" EmbeddedNotificationRequest ")" /
                  "C" "(" EmbeddedModifyConnection ")"

EmbeddedNotificationRequest = ("R" "(" EmbeddedRequestList
                                    ")"


  "," "S" "(" EmbeddedSignalRequest ")" ]
                  "," "D" "(" EmbeddedDigitMap ")" ] )
              / ( "S" "(" EmbeddedSignalRequest ")"
                  "," "D" "(" EmbeddedDigitMap ")" ] )
              / ( "D" "(" EmbeddedDigitMap ")" )

EmbeddedRequestList = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap

SignalRequests = [ SignalRequest 0*("," 0*(WSP)
                   SignalRequest ) ]

SignalRequest = eventName [ "(" signalParameter ")" ]
signalParameter = signalParameterValue /
signalParameterName "=" signalParameterValue /
signalParameterName "(" signalParameterList ")"

signalParameterlist = signalParameterValue 0*("," 0*(WSP)
                     signalParameterValue)

signalParameterName = *(allowed-Characters)
signalParameterValue = 1*(SuitableCharacter)

DigitMap = DigitString  / "(" DigitStringList ")"
DigitStringList = DigitString 0*( "|" DigitString )
DigitString = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition = DigitMapLetter / DigitMapRange
DigitMapLetter = DIGIT / "#" / "*" / "A" / "B" / "C" / "D"
                / "T"/ "a" / "b"/ "c" / "d" /"t"

DigitMapRange =  "x"|"X" / "[" 1*DigitLetter "]"
DigitLetter ::= *((DIGIT "-" DIGIT ) / DigitMapLetter)

ObservedEvents = SignalRequests
EventStates = SignalRequests

ConnectionParameters = [ConnectionParameter
                        0*("," 0*(WSP) ConnectionParameter)
```

```
ConnectionParameter = ( "PS" "=" packetsSent )
                    / ( "OS" "=" octetsSent )
                    / ( "PR" "=" packetsReceived )
                    / ( "OR" "=" octetsReceived )
                    / ( "PL" "=" packetsLost )
                    / ( "JI" "=" jitter )
                    / ( "LA" "=" averageLatency )
                    / ( ConnectionParameterExtensionName
                    "=" ConnectionParameterExtensionValue )

packetsSent = 1*9(DIGIT)
octetsSent = 1*9(DIGIT)
packetsReceived = 1*9(DIGIT)
octetsReceived = 1*9(DIGIT)
packetsLost = 1*9(DIGIT)
jitter = 1*9(DIGIT)
averageLatency = 1*9(DIGIT)
ConnectionParameterExtensionName = "X" "-" 2*ALPHA
ConnectionParameterExtensionValue = 1*9(DIGIT)

ReasonCode = 3DIGIT [SPACE 1*(%x20-7E)]

SpecificEndpointID = endpointName
SecondEndpointID = endpointName

RequestedInfo = [infoCode 0*("," infoCode)]

infoCode = "C" / "I" / "N" / "X" / "L" / "M" / "R" / "S" /
           "D" / "O" / "P" / "E" / "Z" / "Q" / "T" / "RC" /
           "LC" / "A" / "ES" / "RM" / "RD"

QuarantineHandling = "process" / "discard"

DetectEvents = [eventName 0*("," eventName)]

RestartMethod = "graceful" / "forced" / "restart" /
                "disconnected"

RestartDelay = 1*6(DIGIT)

extensionParameter = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
parameterString = 1*(%x20-7F)

MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                [EOL *SDPinformation]

MGCPResponseLine = (<responseCode> 1*(WSP) <transaction-id>
                   [1*(WSP) <responseString>] EOL)
```

```
responseCode = 3DIGIT
responseString = *(%x20-7E)


SuitableCharacter= DIGIT / ALPHA / "+" / "-" / "_" / "&" /
"!" / "'" / "|" / "=" / "#" / "?" / "/" /"." / "$" / "*" / ";"
/ "@" / "[" / "]" /"^" / "`" / "{" / "}" / "~"

quotedString = DQUOTE visibleString
                0*(quoteEscape visibleString) DQUOTE
quoteEscape = DQUOTE DQUOTE
visibleString = (%x00-21 / %x23-FF)
EOL = CRLF / LF
```

SESSION DESCRIPTION PROTOCOL (SDP) GRAMMAR

The following is an Augmented BNF grammar for SDP (reference [4]). ABNF is defined in RFC 2234.

```
announcement = proto-version
               origin-field
               session-name-field
               information-field
               uri-field
               email-fields
               phone-fields
               connection-field
               bandwidth-fields
               time-fields
               key-field
               attribute-fields
               media-descriptions

proto-version = "v=" 1*DIGIT CRLF
                 ;this memo describes version 0

origin-field = "o=" username space
                sess-id space sess-version space
                nettype space addrtype space
                addr CRLF

session-name-field = "s=" text CRLF

information-field = ["i=" text CRLF]
```

```
uri-field = ["u=" uri CRLF]

email-fields = *("e=" email-address CRLF)

phone-fields = *("p=" phone-number CRLF)

connection-field = ["c=" nettype space addrtype space
                     connection-address CRLF]
;a connection field must be present in every media description
or at thesession-level

bandwidth-fields = *("b=" bwtype ":" bandwidth CRLF)

time-fields = 1*( "t=" start-time space stop-time
              *(CRLF repeat-fields) CRLF)
              [zone-adjustments CRLF]

repeat-fields = "r=" repeat-interval space typed-time
                 1*(space typed-time)

zone-adjustments = time space ["-"] typed-time
                   *(space time space ["-"] typed-time)

key-field = ["k=" key-type CRLF]

key-type = "prompt" | "clear:" key-data | "base64:" key-data |
           "uri:" uri

key-data = email-safe | "~" | "

attribute-fields = *("a=" attribute CRLF)

media-descriptions = *( media-field information-field
*(connection-field)   bandwidth-fields   key-field   attribute-
fields
                      )
media-field = "m=" media space port ["/" integer]
              space proto 1*(space fmt) CRLF

media = 1*(alpha-numeric)
        ;typically "audio", "video", "application" or "data"

fmt = 1*(alpha-numeric)
      ;typically an RTP payload type for audio and video media

proto = 1*(alpha-numeric)
        ;typically "RTP/AVP" or "udp" for IP4
```

```
port = 1*(DIGIT)

;should in the range "1024" to "65535" inclusive for UDP based
media

attribute = (att-field ":" att-value) | att-field

att-field = 1*(alpha-numeric)

att-value = byte-string

sess-id = 1*(DIGIT)
          ;should be unique for this originating username/host

sess-version = 1*(DIGIT)
               ;0 is a new session

connection-address = multicast-address | addr

multicast-address = 3*(decimal-uchar ".") decimal-uchar "/"
ttl [ "/" integer ]
                    ;multicast addresses may be in the range
                    ;224.0.0.0 to 239.255.255.255

ttl = decimal-uchar

start-time = time | "0"

stop-time = time | "0"

time = POS-DIGIT 9*(DIGIT)
       ;sufficient for 2 more centuries

repeat-interval = typed-time

typed-time = 1*(DIGIT) [fixed-len-time-unit]

fixed-len-time-unit = "d" | "h" | "m" | "s"

bwtype = 1*(alpha-numeric)

bandwidth = 1*(DIGIT)

username = safe
          ;pretty wide definition, but doesn't include space

email-address = email | email "(" email-safe ")" | email-safe
                "<" email ">"
```

```
email = ;defined in RFC822

uri= ;defined in RFC1630

phone-number = phone | phone "(" email-safe ")" |email-safe
               "<" phone ">"

phone = "+" POS-DIGIT 1*(space | "-" | DIGIT)
        ;there must be a space or hyphen between the
        ;international code and the rest of the number.

nettype = "IN"
          ;list to be extended

addrtype = "IP4" | "IP6"
           ;list to be extended

addr = FQDN | unicast-address

FQDN = alpha-numeric|"-"|".")
       ;fully qualified domain name as specified in RFC1035

unicast-address = IP4-address | IP6-address

IP4-address = b1 "." decimal-uchar "." decimal-uchar "." b4
b1 = decimal-uchar
     ;less than "224"; not "0" or "127"
b4 = decimal-uchar
     ;not "0"
IP6-address = ;to be defined

text = byte-string
       ;default is to interpret this as IS0-10646 UTF8
       ;ISO 8859-1 requires a "a=charset:ISO-8859-1"
       ;session-level attribute to be used

byte-string = 1*(0x01..0x09|0x0b|0x0c|0x0e..0xff)
              ;any byte except NUL, CR or LF


decimal-uchar = DIGIT OS-DIGIT DIGIT
                | ("1" 2*(DIGIT))
                | ("2" ("0"|"1"|"2"|"3"|"4") DIGIT)
                | ("2" "5" ("0"|"1"|"2"|"3"|"4"|"5"))

integer =  POS-DIGIT *(DIGIT)
```

```
alpha-numeric = ALPHA | DIGIT

DIGIT = "0" | POS-DIGIT

POS-DIGIT = "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

ALPHA = "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|
        "n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z"|
        "A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|"K"|"L"|"M"|
        "N"|"O"|"P"|" Q"|"R"|"S"|"T"|"U"|"V"|"W"|"X"|"Y"|"Z"

email-safe = safe | space | tab

safe = alpha-numeric | "'" | "'" | "-" | "." | "/" | ":" | "?"
       | """ | "#" | "$" | "&" | "*" | ";" | "=" | "@" | "[" |
       "]" | "^" | "_" | "`" | "{" | "|" | "}" | "+" |"~" | "

space = %d32
tab = %d9
CRLF = %d13.10
```

References

[1] Bill Douskalis, IP Telephony -The Integration of Robust VoIP Services, Prentice Hall Publication

[2] Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema, and Scott Pickett, *Media Gateway Control Protocol (MGCP) Version 1.*0, RFC 2705, October, 1999. http://www.PacketCable.com/.

[3] Packet Cable *Network-Based Call Signaling (NCS) Protocol* Specification, PKT-SP-EC-MGCP-I03-010620, Interim Release 3.0 June 20, 2001 http://www.PacketCable.com/.

[4] Handley, M, Jacobson, V., *SDP: Session Description Protoco*l, RFC 2327, April 1998.

[5] *PacketCable Security Specificatio*n, PKT-SP-SEC-I03-010620, June 20, 2001,Cable Television Laboratories, Inc., http://www.PacketCable.com/.

[6] http://www.protocols.com

[7] Mauricio Arango, Christian Huitema, *Simple Gateway Control Protocol* 1.1, draft-huitema-mgcp-v1-02.txt , July 30, 1998.

[8] P. Tom Taylor, Pat R. Calhoun, Allan C. Rubens, IPDC Base Protocol, draft-taylor-ipdc-00.txt, July 1998.