DECLARATION OF SANDY GINOZA FOR IETF ADMINISTRATION LLC (IETF)

RFC 3261: Session Initiation Protocol
RFC 3435: Media Gateway Control Protocol
RFC 3660: Basic Media Gateway Control Protocol Packages

I, Sandy Ginoza, hereby declare that all statements made herein are of my own knowledge and are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code:

1. I am an employee of Association Management Solutions, LLC (AMS), which acts under contract to the IETF Administration LLC (IETF) as the operator of the RFC Production Center. The RFC Production Center is part of the "RFC Editor" function, which prepares documents for publication and places files in an online repository for the authoritative Request for Comments (RFC) series of documents (RFC Series), and preserves records relating to these documents. The RFC Series includes, among other things, the series of Internet standards developed by the IETF. I hold the position of Director of the RFC Production Center. I began employment with AMS in this capacity on 6 January 2010.

2. My responsibilities as Director of the RFC Production Center include acting as the custodian of records relating to the RFC Series, and I am familiar with the record keeping practices relating to the RFC Series, including the creation and maintenance of such records.

3. From June 1999 to 5 January 2010, I was an employee of the Information Sciences Institute at University of Southern California (ISI). I held various position titles

with the RFC Editor project at ISI, ending with Senior Editor.

4. The RFC Editor function was conducted by ISI under contract to the United States government prior to 1998. In 1998, the Internet Society (ISOC), in furtherance of its IETF activity, entered into the first in a series of contracts with ISI providing for ISI's performance of the RFC Editor function. Beginning in 2010, certain aspects of the RFC Editor function were assumed by the RFC Production Center operation of AMS under contract to ISOC (acting through its IETF function and, in particular, the IETF Administrative Oversight Committee (now the IETF Administration LLC). The business records of the RFC Editor function, as it was conducted by ISI, are currently housed with a cloud vendor under contract with IETF Administration LLC.

5. I make this declaration based on my personal knowledge and information contained in the business records of the RFC Editor as they are currently housed by AMS with a cloud vendor under contract with IETF Administration LLC, or confirmation with other responsible RFC Editor personnel with such knowledge.

6. Prior to 1998, the RFC Editor's regular practice was to publish RFCs, making them available from a repository via FTP. When a new RFC was published, an announcement of its publication, with information on how to access the RFC, would be typically sent out within 24 hours of the publication.

7. Since 1998, the RFC Editor’s regular practice was to publish RFCs, making them available on the RFC Editor website or via FTP. When a new RFC was published, an announcement of its publication, with information on how to access the RFC, would be typically sent out within 24 hours of the publication. The announcement would go out to all subscribers and a contemporaneous electronic record of the announcement is kept

in the IETF mail archive that is available online.

8. Beginning in 1998, any RFC published on the RFC Editor website or via FTP was reasonably accessible to the public and was disseminated or otherwise available to the extent that persons interested and ordinarily skilled in the subject matter or art exercising reasonable diligence could have located it. In particular, the RFCs were indexed and placed in a public repository.

9. The RFCs are kept in an online repository in the course of the RFC Editor's regularly conducted activity and ordinary course of business. The records are made pursuant to established procedures and are relied upon by the RFC Editor in the performance of its functions.

10. It is the regular practice of the RFC Editor to make and keep the RFC records.

11. Based on the business records for the RFC Editor and the RFC Editor’s course of conduct in publishing RFCs, I have determined that the publication date of RFC 3261 was no later than July, 2002, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. An announcement of its publication also would have been sent out to subscribers within 24 hours of its publication. A copy of that RFC is attached to this declaration as **Exhibit 1**.

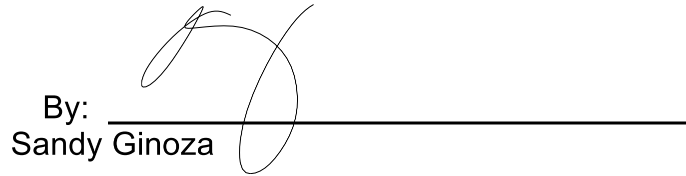
12. Based on the business records for the RFC Editor and the RFC Editor’s course of conduct in publishing RFCs, I have determined that the publication date of RFC 3435 was no later than January, 2003, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. An announcement of its publication also would have been sent out to subscribers within 24 hours of its

publication. A copy of that RFC is attached to this declaration as **Exhibit 2**.

13. Based on the business records for the RFC Editor and the RFC Editor's course of conduct in publishing RFCs, I have determined that the publication date of RFC 3660 was no later than December, 2003, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. An announcement of its publication also would have been sent out to subscribers within 24 hours of its publication. A copy of that RFC is attached to this declaration as **Exhibit 3**.

[REMAINDER OF PAGE LEFT INTENTIONALLY BLANK]

PURSUANT TO SECTION 1746 OF TITLE 28 OF UNITED STATES CODE, I DECLARE UNDER PENALTY OF PERJURY UNDER THE LAWS OF THE UNITED STATES OF AMERICA THAT THE FOREGOING IS TRUE AND CORRECT AND THAT THE FOREGOING IS BASED UPON PERSONAL KNOWLEDGE AND INFORMATION AND IS BELIEVED TO BE TRUE.



Date: 25 November 2024

By: ______________________
Sandy Ginoza

Network Working Group
Request for Comments: 3261
Obsoletes: 2543
Category: Standards Track

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia U.
G. Camarillo
Ericsson
A. Johnston
WorldCom
J. Peterson
Neustar
R. Sparks
dynamicsoft
M. Handley
ICIR
E. Schooler
AT&T

June 2002

SIP: Session Initiation Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

This document describes Session Initiation Protocol (SIP), an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences.

SIP invitations used to create sessions carry session descriptions that allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers. SIP runs on top of several different transport protocols.

Table of Contents

1 Introduction 8
2 Overview of SIP Functionality 9
3 Terminology 10
4 Overview of Operation 10
5 Structure of the Protocol 18
6 Definitions 20
7 SIP Messages 26
7.1 Requests 27
7.2 Responses 28
7.3 Header Fields 29
7.3.1 Header Field Format 30
7.3.2 Header Field Classification 32
7.3.3 Compact Form 32
7.4 Bodies 33
7.4.1 Message Body Type 33
7.4.2 Message Body Length 33
7.5 Framing SIP Messages 34
8 General User Agent Behavior 34
8.1 UAC Behavior 35
8.1.1 Generating the Request 35
8.1.1.1 Request-URI 35
8.1.1.2 To 36
8.1.1.3 From 37
8.1.1.4 Call-ID 37
8.1.1.5 CSeq 38
8.1.1.6 Max-Forwards 38
8.1.1.7 Via 39
8.1.1.8 Contact 40
8.1.1.9 Supported and Require 40
8.1.1.10 Additional Message Components 41
8.1.2 Sending the Request 41
8.1.3 Processing Responses 42
8.1.3.1 Transaction Layer Errors 42
8.1.3.2 Unrecognized Responses 42
8.1.3.3 Vias 43
8.1.3.4 Processing 3xx Responses 43
8.1.3.5 Processing 4xx Responses 45
8.2 UAS Behavior 46
8.2.1 Method Inspection 46
8.2.2 Header Inspection 46
8.2.2.1 To and Request-URI 46
8.2.2.2 Merged Requests 47
8.2.2.3 Require 47
8.2.3 Content Processing 48
8.2.4 Applying Extensions 49
8.2.5 Processing the Request 49

8.2.6 Generating the Response 49
8.2.6.1 Sending a Provisional Response 49
8.2.6.2 Headers and Tags 50
8.2.7 Stateless UAS Behavior 50
8.3 Redirect Servers 51
9 Canceling a Request 53
9.1 Client Behavior 53
9.2 Server Behavior 55
10 Registrations 56
10.1 Overview 56
10.2 Constructing the REGISTER Request 57
10.2.1 Adding Bindings 59
10.2.1.1 Setting the Expiration Interval of Contact Addresses 60
10.2.1.2 Preferences among Contact Addresses 61
10.2.2 Removing Bindings 61
10.2.3 Fetching Bindings 61
10.2.4 Refreshing Bindings 61
10.2.5 Setting the Internal Clock 62
10.2.6 Discovering a Registrar 62
10.2.7 Transmitting a Request 62
10.2.8 Error Responses 63
10.3 Processing REGISTER Requests 63
11 Querying for Capabilities 66
11.1 Construction of OPTIONS Request 67
11.2 Processing of OPTIONS Request 68
12 Dialogs 69
12.1 Creation of a Dialog 70
12.1.1 UAS behavior 70
12.1.2 UAC Behavior 71
12.2 Requests within a Dialog 72
12.2.1 UAC Behavior 73
12.2.1.1 Generating the Request 73
12.2.1.2 Processing the Responses 75
12.2.2 UAS Behavior 76
12.3 Termination of a Dialog 77
13 Initiating a Session 77
13.1 Overview 77
13.2 UAC Processing 78
13.2.1 Creating the Initial INVITE 78
13.2.2 Processing INVITE Responses 81
13.2.2.1 1xx Responses 81
13.2.2.2 3xx Responses 81
13.2.2.3 4xx, 5xx and 6xx Responses 81
13.2.2.4 2xx Responses 82
13.3 UAS Processing 83
13.3.1 Processing of the INVITE 83
13.3.1.1 Progress 84
13.3.1.2 The INVITE is Redirected 84

13.3.1.3 The INVITE is Rejected 85
13.3.1.4 The INVITE is Accepted 85
14 Modifying an Existing Session 86
14.1 UAC Behavior 86
14.2 UAS Behavior 88
15 Terminating a Session 89
15.1 Terminating a Session with a BYE Request 90
15.1.1 UAC Behavior 90
15.1.2 UAS Behavior 91
16 Proxy Behavior 91
16.1 Overview 91
16.2 Stateful Proxy 92
16.3 Request Validation 94
16.4 Route Information Preprocessing 96
16.5 Determining Request Targets 97
16.6 Request Forwarding 99
16.7 Response Processing 107
16.8 Processing Timer C 114
16.9 Handling Transport Errors 115
16.10 CANCEL Processing 115
16.11 Stateless Proxy 116
16.12 Summary of Proxy Route Processing 118
16.12.1 Examples 118
16.12.1.1 Basic SIP Trapezoid 118
16.12.1.2 Traversing a Strict-Routing Proxy 120
16.12.1.3 Rewriting Record-Route Header Field Values 121
17 Transactions 122
17.1 Client Transaction 124
17.1.1 INVITE Client Transaction 125
17.1.1.1 Overview of INVITE Transaction 125
17.1.1.2 Formal Description 125
17.1.1.3 Construction of the ACK Request 129
17.1.2 Non-INVITE Client Transaction 130
17.1.2.1 Overview of the non-INVITE Transaction 130
17.1.2.2 Formal Description 131
17.1.3 Matching Responses to Client Transactions 132
17.1.4 Handling Transport Errors 133
17.2 Server Transaction 134
17.2.1 INVITE Server Transaction 134
17.2.2 Non-INVITE Server Transaction 137
17.2.3 Matching Requests to Server Transactions 138
17.2.4 Handling Transport Errors 141
18 Transport 141
18.1 Clients 142
18.1.1 Sending Requests 142
18.1.2 Receiving Responses 144
18.2 Servers 145
18.2.1 Receiving Requests 145

18.2.2 Sending Responses 146
18.3 Framing 147
18.4 Error Handling 147
19 Common Message Components 147
19.1 SIP and SIPS Uniform Resource Indicators 148
19.1.1 SIP and SIPS URI Components 148
19.1.2 Character Escaping Requirements 152
19.1.3 Example SIP and SIPS URIs 153
19.1.4 URI Comparison 153
19.1.5 Forming Requests from a URI 156
19.1.6 Relating SIP URIs and tel URLs 157
19.2 Option Tags 158
19.3 Tags 159
20 Header Fields 159
20.1 Accept 161
20.2 Accept-Encoding 163
20.3 Accept-Language 164
20.4 Alert-Info 164
20.5 Allow 165
20.6 Authentication-Info 165
20.7 Authorization 165
20.8 Call-ID 166
20.9 Call-Info 166
20.10 Contact 167
20.11 Content-Disposition 168
20.12 Content-Encoding 169
20.13 Content-Language 169
20.14 Content-Length 169
20.15 Content-Type 170
20.16 CSeq 170
20.17 Date 170
20.18 Error-Info 171
20.19 Expires 171
20.20 From 172
20.21 In-Reply-To 172
20.22 Max-Forwards 173
20.23 Min-Expires 173
20.24 MIME-Version 173
20.25 Organization 174
20.26 Priority 174
20.27 Proxy-Authenticate 174
20.28 Proxy-Authorization 175
20.29 Proxy-Require 175
20.30 Record-Route 175
20.31 Reply-To 176
20.32 Require 176
20.33 Retry-After 176
20.34 Route 177

20.35 Server 177
20.36 Subject 177
20.37 Supported 178
20.38 Timestamp 178
20.39 To 178
20.40 Unsupported 179
20.41 User-Agent 179
20.42 Via 179
20.43 Warning 180
20.44 WWW-Authenticate 182
21 Response Codes 182
21.1 Provisional 1xx 182
21.1.1 100 Trying 183
21.1.2 180 Ringing 183
21.1.3 181 Call Is Being Forwarded 183
21.1.4 182 Queued 183
21.1.5 183 Session Progress 183
21.2 Successful 2xx 183
21.2.1 200 OK 183
21.3 Redirection 3xx 184
21.3.1 300 Multiple Choices 184
21.3.2 301 Moved Permanently 184
21.3.3 302 Moved Temporarily 184
21.3.4 305 Use Proxy 185
21.3.5 380 Alternative Service 185
21.4 Request Failure 4xx 185
21.4.1 400 Bad Request 185
21.4.2 401 Unauthorized 185
21.4.3 402 Payment Required 186
21.4.4 403 Forbidden 186
21.4.5 404 Not Found 186
21.4.6 405 Method Not Allowed 186
21.4.7 406 Not Acceptable 186
21.4.8 407 Proxy Authentication Required 186
21.4.9 408 Request Timeout 186
21.4.10 410 Gone 187
21.4.11 413 Request Entity Too Large 187
21.4.12 414 Request-URI Too Long 187
21.4.13 415 Unsupported Media Type 187
21.4.14 416 Unsupported URI Scheme 187
21.4.15 420 Bad Extension 187
21.4.16 421 Extension Required 188
21.4.17 423 Interval Too Brief 188
21.4.18 480 Temporarily Unavailable 188
21.4.19 481 Call/Transaction Does Not Exist 188
21.4.20 482 Loop Detected 188
21.4.21 483 Too Many Hops 189
21.4.22 484 Address Incomplete 189

21.4.23 485 Ambiguous 189
21.4.24 486 Busy Here 189
21.4.25 487 Request Terminated 190
21.4.26 488 Not Acceptable Here 190
21.4.27 491 Request Pending 190
21.4.28 493 Undecipherable 190
21.5 Server Failure 5xx 190
21.5.1 500 Server Internal Error 190
21.5.2 501 Not Implemented 191
21.5.3 502 Bad Gateway 191
21.5.4 503 Service Unavailable 191
21.5.5 504 Server Time-out 191
21.5.6 505 Version Not Supported 192
21.5.7 513 Message Too Large 192
21.6 Global Failures 6xx 192
21.6.1 600 Busy Everywhere 192
21.6.2 603 Decline 192
21.6.3 604 Does Not Exist Anywhere 192
21.6.4 606 Not Acceptable 192
22 Usage of HTTP Authentication 193
22.1 Framework 193
22.2 User-to-User Authentication 195
22.3 Proxy-to-User Authentication 197
22.4 The Digest Authentication Scheme 199
23 S/MIME 201
23.1 S/MIME Certificates 201
23.2 S/MIME Key Exchange 202
23.3 Securing MIME bodies 205
23.4 SIP Header Privacy and Integrity using S/MIME:
Tunneling SIP 207
23.4.1 Integrity and Confidentiality Properties of SIP
Headers 207
23.4.1.1 Integrity 207
23.4.1.2 Confidentiality 208
23.4.2 Tunneling Integrity and Authentication 209
23.4.3 Tunneling Encryption 211
24 Examples 213
24.1 Registration 213
24.2 Session Setup 214
25 Augmented BNF for the SIP Protocol 219
25.1 Basic Rules 219
26 Security Considerations: Threat Model and Security
Usage Recommendations 232
26.1 Attacks and Threat Models 233
26.1.1 Registration Hijacking 233
26.1.2 Impersonating a Server 234
26.1.3 Tampering with Message Bodies 235
26.1.4 Tearing Down Sessions 235

26.1.5 Denial of Service and Amplification 236
26.2 Security Mechanisms 237
26.2.1 Transport and Network Layer Security 238
26.2.2 SIPS URI Scheme 239
26.2.3 HTTP Authentication 240
26.2.4 S/MIME 240
26.3 Implementing Security Mechanisms 241
26.3.1 Requirements for Implementers of SIP 241
26.3.2 Security Solutions 242
26.3.2.1 Registration 242
26.3.2.2 Interdomain Requests 243
26.3.2.3 Peer-to-Peer Requests 245
26.3.2.4 DoS Protection 246
26.4 Limitations 247
26.4.1 HTTP Digest 247
26.4.2 S/MIME 248
26.4.3 TLS 249
26.4.4 SIPS URIs 249
26.5 Privacy 251
27 IANA Considerations 252
27.1 Option Tags 252
27.2 Warn-Codes 252
27.3 Header Field Names 253
27.4 Method and Response Codes 253
27.5 The "message/sip" MIME type. 254
27.6 New Content-Disposition Parameter Registrations 255
28 Changes From RFC 2543 255
28.1 Major Functional Changes 255
28.2 Minor Functional Changes 260
29 Normative References 261
30 Informative References 262
A Table of Timer Values 265
Acknowledgments 266
Authors' Addresses 267
Full Copyright Statement 269

1 Introduction

There are many applications of the Internet that require the creation and management of a session, where a session is considered an exchange of data between an association of participants. The implementation of these applications is complicated by the practices of participants: users may move between endpoints, they may be addressable by multiple names, and they may communicate in several different media - sometimes simultaneously. Numerous protocols have been authored that carry various forms of real-time multimedia session data such as voice, video, or text messages. The Session Initiation Protocol (SIP) works in concert with these protocols by

enabling Internet endpoints (called user agents) to discover one another and to agree on a characterization of a session they would like to share. For locating prospective session participants, and for other functions, SIP enables the creation of an infrastructure of network hosts (called proxy servers) to which user agents can send registrations, invitations to sessions, and other requests. SIP is an agile, general-purpose tool for creating, modifying, and terminating sessions that works independently of underlying transport protocols and without dependency on the type of session that is being established.

2 Overview of SIP Functionality

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants to already existing sessions, such as multicast conferences. Media can be added to (and removed from) an existing session. SIP transparently supports name mapping and redirection services, which supports personal mobility [27] - users can maintain a single externally visible identifier regardless of their network location.

SIP supports five facets of establishing and terminating multimedia communications:

User location: determination of the end system to be used for communication;

User availability: determination of the willingness of the called party to engage in communications;

User capabilities: determination of the media and media parameters to be used;

Session setup: "ringing", establishment of session parameters at both called and calling party;

Session management: including transfer and termination of sessions, modifying session parameters, and invoking services.

SIP is not a vertically integrated communications system. SIP is rather a component that can be used with other IETF protocols to build a complete multimedia architecture. Typically, these architectures will include protocols such as the Real-time Transport Protocol (RTP) (RFC 1889 [28]) for transporting real-time data and providing QoS feedback, the Real-Time streaming protocol (RTSP) (RFC 2326 [29]) for controlling delivery of streaming media, the Media

Gateway Control Protocol (MEGACO) (RFC 3015 [30]) for controlling gateways to the Public Switched Telephone Network (PSTN), and the Session Description Protocol (SDP) (RFC 2327 [1]) for describing multimedia sessions. Therefore, SIP should be used in conjunction with other protocols in order to provide complete services to the users. However, the basic functionality and operation of SIP does not depend on any of these protocols.

SIP does not provide services. Rather, SIP provides primitives that can be used to implement different services. For example, SIP can locate a user and deliver an opaque object to his current location. If this primitive is used to deliver a session description written in SDP, for instance, the endpoints can agree on the parameters of a session. If the same primitive is used to deliver a photo of the caller as well as the session description, a "caller ID" service can be easily implemented. As this example shows, a single primitive is typically used to provide several different services.

SIP does not offer conference control services such as floor control or voting and does not prescribe how a conference is to be managed. SIP can be used to initiate a session that uses some other conference control protocol. Since SIP messages and the sessions they establish can pass through entirely different networks, SIP cannot, and does not, provide any kind of network resource reservation capabilities.

The nature of the services provided make security particularly important. To that end, SIP provides a suite of security services, which include denial-of-service prevention, authentication (both user to user and proxy to user), integrity protection, and encryption and privacy services.

SIP works with both IPv4 and IPv6.

3 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [2] and indicate requirement levels for compliant SIP implementations.

4 Overview of Operation

This section introduces the basic operations of SIP using simple examples. This section is tutorial in nature and does not contain any normative statements.

The first example shows the basic functions of SIP: location of an end point, signal of a desire to communicate, negotiation of session parameters to establish the session, and teardown of the session once established.

Figure 1 shows a typical example of a SIP message exchange between two users, Alice and Bob. (Each message is labeled with the letter "F" and a number for reference by the text.) In this example, Alice uses a SIP application on her PC (referred to as a softphone) to call Bob on his SIP phone over the Internet. Also shown are two SIP proxy servers that act on behalf of Alice and Bob to facilitate the session establishment. This typical arrangement is often referred to as the "SIP trapezoid" as shown by the geometric shape of the dotted lines in Figure 1.

Alice "calls" Bob using his SIP identity, a type of Uniform Resource Identifier (URI) called a SIP URI. SIP URIs are defined in Section 19.1. It has a similar form to an email address, typically containing a username and a host name. In this case, it is sip:bob@biloxi.com, where biloxi.com is the domain of Bob's SIP service provider. Alice has a SIP URI of sip:alice@atlanta.com. Alice might have typed in Bob's URI or perhaps clicked on a hyperlink or an entry in an address book. SIP also provides a secure URI, called a SIPS URI. An example would be sips:bob@biloxi.com. A call made to a SIPS URI guarantees that secure, encrypted transport (namely TLS) is used to carry all SIP messages from the caller to the domain of the callee. From there, the request is sent securely to the callee, but with security mechanisms that depend on the policy of the domain of the callee.

SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a request that invokes a particular method, or function, on the server and at least one response. In this example, the transaction begins with Alice's softphone sending an INVITE request addressed to Bob's SIP URI. INVITE is an example of a SIP method that specifies the action that the requestor (Alice) wants the server (Bob) to take. The INVITE request contains a number of header fields. Header fields are named attributes that provide additional information about a message. The ones present in an INVITE include a unique identifier for the call, the destination address, Alice's address, and information about the type of session that Alice wishes to establish with Bob. The INVITE (message F1 in Figure 1) might look like this:

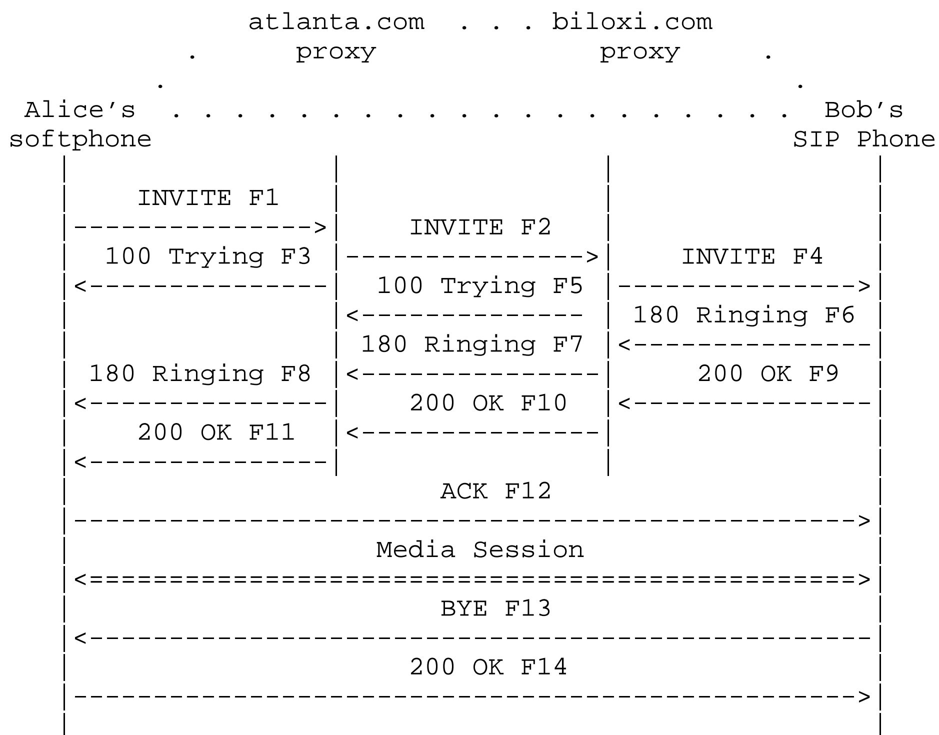


Figure 1: SIP session setup example with SIP trapezoid

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

The first line of the text-encoded message contains the method name (INVITE). The lines that follow are a list of header fields. This example contains a minimum required set. The header fields are briefly described below:

Via contains the address (pc33.atlanta.com) at which Alice is expecting to receive responses to this request. It also contains a branch parameter that identifies this transaction.

To contains a display name (Bob) and a SIP or SIPS URI (sip:bob@biloxi.com) towards which the request was originally directed. Display names are described in RFC 2822 [3].

From also contains a display name (Alice) and a SIP or SIPS URI (sip:alice@atlanta.com) that indicate the originator of the request. This header field also has a tag parameter containing a random string (1928301774) that was added to the URI by the softphone. It is used for identification purposes.

Call-ID contains a globally unique identifier for this call, generated by the combination of a random string and the softphone's host name or IP address. The combination of the To tag, From tag, and Call-ID completely defines a peer-to-peer SIP relationship between Alice and Bob and is referred to as a dialog.

CSeq or Command Sequence contains an integer and a method name. The CSeq number is incremented for each new request within a dialog and is a traditional sequence number.

Contact contains a SIP or SIPS URI that represents a direct route to contact Alice, usually composed of a username at a fully qualified domain name (FQDN). While an FQDN is preferred, many end systems do not have registered domain names, so IP addresses are permitted. While the Via header field tells other elements where to send the response, the Contact header field tells other elements where to send future requests.

Max-Forwards serves to limit the number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one at each hop.

Content-Type contains a description of the message body (not shown).

Content-Length contains an octet (byte) count of the message body.

The complete set of SIP header fields is defined in Section 20.

The details of the session, such as the type of media, codec, or sampling rate, are not described using SIP. Rather, the body of a SIP message contains a description of the session, encoded in some other protocol format. One such format is the Session Description Protocol (SDP) (RFC 2327 [1]). This SDP message (not shown in the

example) is carried by the SIP message in a way that is analogous to a document attachment being carried by an email message, or a web page being carried in an HTTP message.

Since the softphone does not know the location of Bob or the SIP server in the biloxi.com domain, the softphone sends the INVITE to the SIP server that serves Alice's domain, atlanta.com. The address of the atlanta.com SIP server could have been configured in Alice's softphone, or it could have been discovered by DHCP, for example.

The atlanta.com SIP server is a type of SIP server known as a proxy server. A proxy server receives SIP requests and forwards them on behalf of the requestor. In this example, the proxy server receives the INVITE request and sends a 100 (Trying) response back to Alice's softphone. The 100 (Trying) response indicates that the INVITE has been received and that the proxy is working on her behalf to route the INVITE to the destination. Responses in SIP use a three-digit code followed by a descriptive phrase. This response contains the same To, From, Call-ID, CSeq and branch parameter in the Via as the INVITE, which allows Alice's softphone to correlate this response to the sent INVITE. The atlanta.com proxy server locates the proxy server at biloxi.com, possibly by performing a particular type of DNS (Domain Name Service) lookup to find the SIP server that serves the biloxi.com domain. This is described in [4]. As a result, it obtains the IP address of the biloxi.com proxy server and forwards, or proxies, the INVITE request there. Before forwarding the request, the atlanta.com proxy server adds an additional Via header field value that contains its own address (the INVITE already contains Alice's address in the first Via). The biloxi.com proxy server receives the INVITE and responds with a 100 (Trying) response back to the atlanta.com proxy server to indicate that it has received the INVITE and is processing the request. The proxy server consults a database, generically called a location service, that contains the current IP address of Bob. (We shall see in the next section how this database can be populated.) The biloxi.com proxy server adds another Via header field value with its own address to the INVITE and proxies it to Bob's SIP phone.

Bob's SIP phone receives the INVITE and alerts Bob to the incoming call from Alice so that Bob can decide whether to answer the call, that is, Bob's phone rings. Bob's SIP phone indicates this in a 180 (Ringing) response, which is routed back through the two proxies in the reverse direction. Each proxy uses the Via header field to determine where to send the response and removes its own address from the top. As a result, although DNS and location service lookups were required to route the initial INVITE, the 180 (Ringing) response can be returned to the caller without lookups or without state being

maintained in the proxies. This also has the desirable property that each proxy that sees the INVITE will also see all responses to the INVITE.

When Alice's softphone receives the 180 (Ringing) response, it passes this information to Alice, perhaps using an audio ringback tone or by displaying a message on Alice's screen.

In this example, Bob decides to answer the call. When he picks up the handset, his SIP phone sends a 200 (OK) response to indicate that the call has been answered. The 200 (OK) contains a message body with the SDP media description of the type of session that Bob is willing to establish with Alice. As a result, there is a two-phase exchange of SDP messages: Alice sent one to Bob, and Bob sent one back to Alice. This two-phase exchange provides basic negotiation capabilities and is based on a simple offer/answer model of SDP exchange. If Bob did not wish to answer the call or was busy on another call, an error response would have been sent instead of the 200 (OK), which would have resulted in no media session being established. The complete list of SIP response codes is in Section 21. The 200 (OK) (message F9 in Figure 1) might look like this as Bob sends it out:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com
   ;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
   ;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com
   ;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)
```

The first line of the response contains the response code (200) and the reason phrase (OK). The remaining lines contain header fields. The Via, To, From, Call-ID, and CSeq header fields are copied from the INVITE request. (There are three Via header field values - one added by Alice's SIP phone, one added by the atlanta.com proxy, and one added by the biloxi.com proxy.) Bob's SIP phone has added a tag parameter to the To header field. This tag will be incorporated by both endpoints into the dialog and will be included in all future

requests and responses in this call. The Contact header field contains a URI at which Bob can be directly reached at his SIP phone. The Content-Type and Content-Length refer to the message body (not shown) that contains Bob's SDP media information.

In addition to DNS and location service lookups shown in this example, proxy servers can make flexible "routing decisions" to decide where to send a request. For example, if Bob's SIP phone returned a 486 (Busy Here) response, the biloxi.com proxy server could proxy the INVITE to Bob's voicemail server. A proxy server can also send an INVITE to a number of locations at the same time. This type of parallel search is known as forking.

In this case, the 200 (OK) is routed back through the two proxies and is received by Alice's softphone, which then stops the ringback tone and indicates that the call has been answered. Finally, Alice's softphone sends an acknowledgement message, ACK, to Bob's SIP phone to confirm the reception of the final response (200 (OK)). In this example, the ACK is sent directly from Alice's softphone to Bob's SIP phone, bypassing the two proxies. This occurs because the endpoints have learned each other's address from the Contact header fields through the INVITE/200 (OK) exchange, which was not known when the initial INVITE was sent. The lookups performed by the two proxies are no longer needed, so the proxies drop out of the call flow. This completes the INVITE/200/ACK three-way handshake used to establish SIP sessions. Full details on session setup are in Section 13.

Alice and Bob's media session has now begun, and they send media packets using the format to which they agreed in the exchange of SDP. In general, the end-to-end media packets take a different path from the SIP signaling messages.

During the session, either Alice or Bob may decide to change the characteristics of the media session. This is accomplished by sending a re-INVITE containing a new media description. This re-INVITE references the existing dialog so that the other party knows that it is to modify an existing session instead of establishing a new session. The other party sends a 200 (OK) to accept the change. The requestor responds to the 200 (OK) with an ACK. If the other party does not accept the change, he sends an error response such as 488 (Not Acceptable Here), which also receives an ACK. However, the failure of the re-INVITE does not cause the existing call to fail - the session continues using the previously negotiated characteristics. Full details on session modification are in Section 14.

At the end of the call, Bob disconnects (hangs up) first and generates a BYE message. This BYE is routed directly to Alice's softphone, again bypassing the proxies. Alice confirms receipt of the BYE with a 200 (OK) response, which terminates the session and the BYE transaction. No ACK is sent - an ACK is only sent in response to a response to an INVITE request. The reasons for this special handling for INVITE will be discussed later, but relate to the reliability mechanisms in SIP, the length of time it can take for a ringing phone to be answered, and forking. For this reason, request handling in SIP is often classified as either INVITE or non-INVITE, referring to all other methods besides INVITE. Full details on session termination are in Section 15.

Section 24.2 describes the messages shown in Figure 1 in full.

In some cases, it may be useful for proxies in the SIP signaling path to see all the messaging between the endpoints for the duration of the session. For example, if the biloxi.com proxy server wished to remain in the SIP messaging path beyond the initial INVITE, it would add to the INVITE a required routing header field known as Record-Route that contained a URI resolving to the hostname or IP address of the proxy. This information would be received by both Bob's SIP phone and (due to the Record-Route header field being passed back in the 200 (OK)) Alice's softphone and stored for the duration of the dialog. The biloxi.com proxy server would then receive and proxy the ACK, BYE, and 200 (OK) to the BYE. Each proxy can independently decide to receive subsequent messages, and those messages will pass through all proxies that elect to receive it. This capability is frequently used for proxies that are providing mid-call features.

Registration is another common operation in SIP. Registration is one way that the biloxi.com server can learn the current location of Bob. Upon initialization, and at periodic intervals, Bob's SIP phone sends REGISTER messages to a server in the biloxi.com domain known as a SIP registrar. The REGISTER messages associate Bob's SIP or SIPS URI (sip:bob@biloxi.com) with the machine into which he is currently logged (conveyed as a SIP or SIPS URI in the Contact header field). The registrar writes this association, also called a binding, to a database, called the location service, where it can be used by the proxy in the biloxi.com domain. Often, a registrar server for a domain is co-located with the proxy for that domain. It is an important concept that the distinction between types of SIP servers is logical, not physical.

Bob is not limited to registering from a single device. For example, both his SIP phone at home and the one in the office could send registrations. This information is stored together in the location

service and allows a proxy to perform various types of searches to locate Bob. Similarly, more than one user can be registered on a single device at the same time.

The location service is just an abstract concept. It generally contains information that allows a proxy to input a URI and receive a set of zero or more URIs that tell the proxy where to send the request. Registrations are one way to create this information, but not the only way. Arbitrary mapping functions can be configured at the discretion of the administrator.

Finally, it is important to note that in SIP, registration is used for routing incoming SIP requests and has no role in authorizing outgoing requests. Authorization and authentication are handled in SIP either on a request-by-request basis with a challenge/response mechanism, or by using a lower layer scheme as discussed in Section 26.

The complete set of SIP message details for this registration example is in Section 24.1.

Additional operations in SIP, such as querying for the capabilities of a SIP server or client using OPTIONS, or canceling a pending request using CANCEL, will be introduced in later sections.

5 Structure of the Protocol

SIP is structured as a layered protocol, which means that its behavior is described in terms of a set of fairly independent processing stages with only a loose coupling between each stage. The protocol behavior is described as layers for the purpose of presentation, allowing the description of functions common across elements in a single section. It does not dictate an implementation in any way. When we say that an element "contains" a layer, we mean it is compliant to the set of rules defined by that layer.

Not every element specified by the protocol contains every layer. Furthermore, the elements specified by SIP are logical elements, not physical ones. A physical realization can choose to act as different logical elements, perhaps even on a transaction-by-transaction basis.

The lowest layer of SIP is its syntax and encoding. Its encoding is specified using an augmented Backus-Naur Form grammar (BNF). The complete BNF is specified in Section 25; an overview of a SIP message's structure can be found in Section 7.

The second layer is the transport layer. It defines how a client sends requests and receives responses and how a server receives requests and sends responses over the network. All SIP elements contain a transport layer. The transport layer is described in Section 18.

The third layer is the transaction layer. Transactions are a fundamental component of SIP. A transaction is a request sent by a client transaction (using the transport layer) to a server transaction, along with all responses to that request sent from the server transaction back to the client. The transaction layer handles application-layer retransmissions, matching of responses to requests, and application-layer timeouts. Any task that a user agent client (UAC) accomplishes takes place using a series of transactions. Discussion of transactions can be found in Section 17. User agents contain a transaction layer, as do stateful proxies. Stateless proxies do not contain a transaction layer. The transaction layer has a client component (referred to as a client transaction) and a server component (referred to as a server transaction), each of which are represented by a finite state machine that is constructed to process a particular request.

The layer above the transaction layer is called the transaction user (TU). Each of the SIP entities, except the stateless proxy, is a transaction user. When a TU wishes to send a request, it creates a client transaction instance and passes it the request along with the destination IP address, port, and transport to which to send the request. A TU that creates a client transaction can also cancel it. When a client cancels a transaction, it requests that the server stop further processing, revert to the state that existed before the transaction was initiated, and generate a specific error response to that transaction. This is done with a CANCEL request, which constitutes its own transaction, but references the transaction to be cancelled (Section 9).

The SIP elements, that is, user agent clients and servers, stateless and stateful proxies and registrars, contain a core that distinguishes them from each other. Cores, except for the stateless proxy, are transaction users. While the behavior of the UAC and UAS cores depends on the method, there are some common rules for all methods (Section 8). For a UAC, these rules govern the construction of a request; for a UAS, they govern the processing of a request and generating a response. Since registrations play an important role in SIP, a UAS that handles a REGISTER is given the special name registrar. Section 10 describes UAC and UAS core behavior for the REGISTER method. Section 11 describes UAC and UAS core behavior for the OPTIONS method, used for determining the capabilities of a UA.

Certain other requests are sent within a dialog. A dialog is a peer-to-peer SIP relationship between two user agents that persists for some time. The dialog facilitates sequencing of messages and proper routing of requests between the user agents. The INVITE method is the only way defined in this specification to establish a dialog. When a UAC sends a request that is within the context of a dialog, it follows the common UAC rules as discussed in Section 8 but also the rules for mid-dialog requests. Section 12 discusses dialogs and presents the procedures for their construction and maintenance, in addition to construction of requests within a dialog.

The most important method in SIP is the INVITE method, which is used to establish a session between participants. A session is a collection of participants, and streams of media between them, for the purposes of communication. Section 13 discusses how sessions are initiated, resulting in one or more SIP dialogs. Section 14 discusses how characteristics of that session are modified through the use of an INVITE request within a dialog. Finally, section 15 discusses how a session is terminated.

The procedures of Sections 8, 10, 11, 12, 13, 14, and 15 deal entirely with the UA core (Section 9 describes cancellation, which applies to both UA core and proxy core). Section 16 discusses the proxy element, which facilitates routing of messages between user agents.

6 Definitions

The following terms have special significance for SIP.

Address-of-Record: An address-of-record (AOR) is a SIP or SIPS URI that points to a domain with a location service that can map the URI to another URI where the user might be available. Typically, the location service is populated through registrations. An AOR is frequently thought of as the "public address" of the user.

Back-to-Back User Agent: A back-to-back user agent (B2BUA) is a logical entity that receives a request and processes it as a user agent server (UAS). In order to determine how the request should be answered, it acts as a user agent client (UAC) and generates requests. Unlike a proxy server, it maintains dialog state and must participate in all requests sent on the dialogs it has established. Since it is a concatenation of a UAC and UAS, no explicit definitions are needed for its behavior.

Call: A call is an informal term that refers to some communication between peers, generally set up for the purposes of a multimedia conversation.

Call Leg: Another name for a dialog [31]; no longer used in this specification.

Call Stateful: A proxy is call stateful if it retains state for a dialog from the initiating INVITE to the terminating BYE request. A call stateful proxy is always transaction stateful, but the converse is not necessarily true.

Client: A client is any network element that sends SIP requests and receives SIP responses. Clients may or may not interact directly with a human user. User agent clients and proxies are clients.

Conference: A multimedia session (see below) that contains multiple participants.

Core: Core designates the functions specific to a particular type of SIP entity, i.e., specific to either a stateful or stateless proxy, a user agent or registrar. All cores, except those for the stateless proxy, are transaction users.

Dialog: A dialog is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. A dialog is identified by a call identifier, local tag, and a remote tag. A dialog was formerly known as a call leg in RFC 2543.

Downstream: A direction of message forwarding within a transaction that refers to the direction that requests flow from the user agent client to user agent server.

Final Response: A response that terminates a SIP transaction, as opposed to a provisional response that does not. All 2xx, 3xx, 4xx, 5xx and 6xx responses are final.

Header: A header is a component of a SIP message that conveys information about the message. It is structured as a sequence of header fields.

Header Field: A header field is a component of the SIP message header. A header field can appear as one or more header field rows. Header field rows consist of a header field name and zero or more header field values. Multiple header field values on a

given header field row are separated by commas. Some header fields can only have a single header field value, and as a result, always appear as a single header field row.

Header Field Value: A header field value is a single value; a header field consists of zero or more header field values.

Home Domain: The domain providing service to a SIP user. Typically, this is the domain present in the URI in the address-of-record of a registration.

Informational Response: Same as a provisional response.

Initiator, Calling Party, Caller: The party initiating a session (and dialog) with an INVITE request. A caller retains this role from the time it sends the initial INVITE that established a dialog until the termination of that dialog.

Invitation: An INVITE request.

Invitee, Invited User, Called Party, Callee: The party that receives an INVITE request for the purpose of establishing a new session. A callee retains this role from the time it receives the INVITE until the termination of the dialog established by that INVITE.

Location Service: A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). It contains a list of bindings of address-of-record keys to zero or more contact addresses. The bindings can be created and removed in many ways; this specification defines a REGISTER method that updates the bindings.

Loop: A request that arrives at a proxy, is forwarded, and later arrives back at the same proxy. When it arrives the second time, its Request-URI is identical to the first time, and other header fields that affect proxy operation are unchanged, so that the proxy would make the same processing decision on the request it made the first time. Looped requests are errors, and the procedures for detecting them and handling them are described by the protocol.

Loose Routing: A proxy is said to be loose routing if it follows the procedures defined in this specification for processing of the Route header field. These procedures separate the destination of the request (present in the Request-URI) from

the set of proxies that need to be visited along the way (present in the Route header field). A proxy compliant to these mechanisms is also known as a loose router.

Message: Data sent between SIP elements as part of the protocol. SIP messages are either requests or responses.

Method: The method is the primary function that a request is meant to invoke on a server. The method is carried in the request message itself. Example methods are INVITE and BYE.

Outbound Proxy: A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols.

Parallel Search: In a parallel search, a proxy issues several requests to possible user locations upon receiving an incoming request. Rather than issuing one request and then waiting for the final response before issuing the next request as in a sequential search, a parallel search issues requests without waiting for the result of previous requests.

Provisional Response: A response used by the server to indicate progress, but that does not terminate a SIP transaction. 1xx responses are provisional, other responses are considered final.

Proxy, Proxy Server: An intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity "closer" to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.

Recursion: A client recurses on a 3xx response when it generates a new request to one or more of the URIs in the Contact header field in the response.

Redirect Server: A redirect server is a user agent server that generates 3xx responses to requests it receives, directing the client to contact an alternate set of URIs.

Registrar: A registrar is a server that accepts REGISTER requests and places the information it receives in those requests into the location service for the domain it handles.

Regular Transaction: A regular transaction is any transaction with a method other than INVITE, ACK, or CANCEL.

Request: A SIP message sent from a client to a server, for the purpose of invoking a particular operation.

Response: A SIP message sent from a server to a client, for indicating the status of a request sent from the client to the server.

Ringback: Ringback is the signaling tone produced by the calling party's application indicating that a called party is being alerted (ringing).

Route Set: A route set is a collection of ordered SIP or SIPS URI which represent a list of proxies that must be traversed when sending a particular request. A route set can be learned, through headers like Record-Route, or it can be configured.

Server: A server is a network element that receives requests in order to service them and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars.

Sequential Search: In a sequential search, a proxy server attempts each contact address in sequence, proceeding to the next one only after the previous has generated a final response. A 2xx or 6xx class final response always terminates a sequential search.

Session: From the SDP specification: "A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session." (RFC 2327 [1]) (A session as defined for SDP can comprise one or more RTP sessions.) As defined, a callee can be invited several times, by different calls, to the same session. If SDP is used, a session is defined by the concatenation of the SDP user name, session id, network type, address type, and address elements in the origin field.

SIP Transaction: A SIP transaction occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to a final (non-1xx) response

sent from the server to the client. If the request is INVITE and the final response is a non-2xx, the transaction also includes an ACK to the response. The ACK for a 2xx response to an INVITE request is a separate transaction.

Spiral: A spiral is a SIP request that is routed to a proxy, forwarded onwards, and arrives once again at that proxy, but this time differs in a way that will result in a different processing decision than the original request. Typically, this means that the request's Request-URI differs from its previous arrival. A spiral is not an error condition, unlike a loop. A typical cause for this is call forwarding. A user calls joe@example.com. The example.com proxy forwards it to Joe's PC, which in turn, forwards it to bob@example.com. This request is proxied back to the example.com proxy. However, this is not a loop. Since the request is targeted at a different user, it is considered a spiral, and is a valid condition.

Stateful Proxy: A logical entity that maintains the client and server transaction state machines defined by this specification during the processing of a request, also known as a transaction stateful proxy. The behavior of a stateful proxy is further defined in Section 16. A (transaction) stateful proxy is not the same as a call stateful proxy.

Stateless Proxy: A logical entity that does not maintain the client or server transaction state machines defined in this specification when it processes requests. A stateless proxy forwards every request it receives downstream and every response it receives upstream.

Strict Routing: A proxy is said to be strict routing if it follows the Route processing rules of RFC 2543 and many prior work in progress versions of this RFC. That rule caused proxies to destroy the contents of the Request-URI when a Route header field was present. Strict routing behavior is not used in this specification, in favor of a loose routing behavior. Proxies that perform strict routing are also known as strict routers.

Target Refresh Request: A target refresh request sent within a dialog is defined as a request that can modify the remote target of the dialog.

Transaction User (TU): The layer of protocol processing that resides above the transaction layer. Transaction users include the UAC core, UAS core, and proxy core.

Upstream: A direction of message forwarding within a transaction that refers to the direction that responses flow from the user agent server back to the user agent client.

URL-encoded: A character string encoded according to RFC 2396, Section 2.4 [5].

User Agent Client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.

UAC Core: The set of processing functions required of a UAC that reside above the transaction and transport layers.

User Agent Server (UAS): A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction.

UAS Core: The set of processing functions required at a UAS that resides above the transaction and transport layers.

User Agent (UA): A logical entity that can act as both a user agent client and user agent server.

The role of UAC and UAS, as well as proxy and redirect servers, are defined on a transaction-by-transaction basis. For example, the user agent initiating a call acts as a UAC when sending the initial INVITE request and as a UAS when receiving a BYE request from the callee. Similarly, the same software can act as a proxy server for one request and as a redirect server for the next request.

Proxy, location, and registrar servers defined above are logical entities; implementations MAY combine them into a single application.

7 SIP Messages

SIP is a text-based protocol and uses the UTF-8 charset (RFC 2279 [7]).

A SIP message is either a request from a client to a server, or a response from a server to a client.

Both Request (section 7.1) and Response (section 7.2) messages use the basic format of RFC 2822 [3], even though the syntax differs in character set and syntax specifics. (SIP allows header fields that would not be valid RFC 2822 header fields, for example.) Both types of messages consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body.

```
generic-message  =  start-line
                    *message-header
                    CRLF
                    [ message-body ]
start-line       =  Request-Line / Status-Line
```

The start-line, each message-header line, and the empty line MUST be terminated by a carriage-return line-feed sequence (CRLF). Note that the empty line MUST be present even if the message-body is not.

Except for the above difference in character sets, much of SIP's message and header field syntax is identical to HTTP/1.1. Rather than repeating the syntax and semantics here, we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification (RFC 2616 [8]).

However, SIP is not an extension of HTTP.

7.1 Requests

SIP requests are distinguished by having a Request-Line for a start-line. A Request-Line contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character.

The Request-Line ends with CRLF. No CR or LF are allowed except in the end-of-line CRLF sequence. No linear whitespace (LWS) is allowed in any of the elements.

```
Request-Line  =  Method SP Request-URI SP SIP-Version CRLF
```

Method: This specification defines six methods: REGISTER for registering contact information, INVITE, ACK, and CANCEL for setting up sessions, BYE for terminating sessions, and OPTIONS for querying servers about their capabilities. SIP extensions, documented in standards track RFCs, may define additional methods.

Request-URI: The Request-URI is a SIP or SIPS URI as described in Section 19.1 or a general URI (RFC 2396 [5]). It indicates the user or service to which this request is being addressed. The Request-URI MUST NOT contain unescaped spaces or control characters and MUST NOT be enclosed in "<>".

SIP elements MAY support Request-URIs with schemes other than "sip" and "sips", for example the "tel" URI scheme of RFC 2806 [9]. SIP elements MAY translate non-SIP URIs using any mechanism at their disposal, resulting in SIP URI, SIPS URI, or some other scheme.

SIP-Version: Both request and response messages include the version of SIP in use, and follow [H3.1] (with HTTP replaced by SIP, and HTTP/1.1 replaced by SIP/2.0) regarding version ordering, compliance requirements, and upgrading of version numbers. To be compliant with this specification, applications sending SIP messages MUST include a SIP-Version of "SIP/2.0". The SIP-Version string is case-insensitive, but implementations MUST send upper-case.

Unlike HTTP/1.1, SIP treats the version number as a literal string. In practice, this should make no difference.

7.2 Responses

SIP responses are distinguished from requests by having a Status-Line as their start-line. A Status-Line consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by a single SP character.

No CR or LF is allowed except in the final CRLF sequence.

```
Status-Line  =  SIP-Version SP Status-Code SP Reason-Phrase CRLF
```

The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. A client is not required to examine or display the Reason-Phrase.

While this specification suggests specific wording for the reason phrase, implementations MAY choose other text, for example, in the language indicated in the Accept-Language header field of the request.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. For this reason, any response with a status code between 100 and 199 is referred to as a "1xx response", any response with a status code between 200 and 299 as a "2xx response", and so on. SIP/2.0 allows six values for the first digit:

1xx: Provisional -- request received, continuing to process the request;

2xx: Success -- the action was successfully received, understood, and accepted;

3xx: Redirection -- further action needs to be taken in order to complete the request;

4xx: Client Error -- the request contains bad syntax or cannot be fulfilled at this server;

5xx: Server Error -- the server failed to fulfill an apparently valid request;

6xx: Global Failure -- the request cannot be fulfilled at any server.

Section 21 defines these classes and describes the individual codes.

7.3 Header Fields

SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header fields follow the [H4.2] definitions of syntax for the message-header and the rules for extending header fields over multiple lines. However, the latter is specified in HTTP with implicit whitespace and folding. This specification conforms to RFC 2234 [10] and uses only explicit whitespace and folding as an integral part of the grammar.

[H4.2] also specifies that multiple header fields of the same field name whose value is a comma-separated list can be combined into one header field. That applies to SIP as well, but the specific rule is different because of the different grammars. Specifically, any SIP header whose grammar is of the form

```
header  =  "header-name" HCOLON header-value *(COMMA header-value)
```

allows for combining header fields of the same name into a comma-separated list. The Contact header field allows a comma-separated list unless the header field value is "*".

7.3.1 Header Field Format

Header fields follow the same generic header format as that given in Section 2.2 of RFC 2822 [3]. Each header field consists of a field name followed by a colon (":") and the field value.

```
field-name: field-value
```

The formal grammar for a message-header specified in Section 25 allows for an arbitrary amount of whitespace on either side of the colon; however, implementations should avoid spaces between the field name and the colon and use a single space (SP) between the colon and the field-value.

```
Subject:            lunch
Subject      :      lunch
Subject            :lunch
Subject: lunch
```

Thus, the above are all valid and equivalent, but the last is the preferred form.

Header fields can be extended over multiple lines by preceding each extra line with at least one SP or horizontal tab (HT). The line break and the whitespace at the beginning of the next line are treated as a single SP character. Thus, the following are equivalent:

```
Subject: I know you're there, pick up the phone and talk to me!
Subject: I know you're there,
         pick up the phone
         and talk to me!
```

The relative order of header fields with different field names is not significant. However, it is RECOMMENDED that header fields which are needed for proxy processing (Via, Route, Record-Route, Proxy-Require, Max-Forwards, and Proxy-Authorization, for example) appear towards the top of the message to facilitate rapid parsing. The relative order of header field rows with the same field name is important. Multiple header field rows with the same field-name MAY be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list (that is, if follows the grammar defined in Section 7.3). It MUST be possible to combine the multiple header field rows into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The exceptions to this rule are the WWW-Authenticate, Authorization, Proxy-Authenticate, and Proxy-Authorization header fields. Multiple header

field rows with these names MAY be present in a message, but since their grammar does not follow the general form listed in Section 7.3, they MUST NOT be combined into a single header field row.

Implementations MUST be able to process multiple header field rows with the same name in any combination of the single-value-per-line or comma-separated value forms.

The following groups of header field rows are valid and equivalent:

```
Route: <sip:alice@atlanta.com>
Subject: Lunch
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>
Subject: Lunch

Subject: Lunch
Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>,
       <sip:carol@chicago.com>
```

Each of the following blocks is valid but not equivalent to the others:

```
Route: <sip:alice@atlanta.com>
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:bob@biloxi.com>
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>,<sip:carol@chicago.com>,
       <sip:bob@biloxi.com>
```

The format of a header field-value is defined per header-name. It will always be either an opaque sequence of TEXT-UTF8 octets, or a combination of whitespace, tokens, separators, and quoted strings. Many existing header fields will adhere to the general form of a value followed by a semi-colon separated sequence of parameter-name, parameter-value pairs:

```
field-name: field-value *(;parameter-name=parameter-value)
```

Even though an arbitrary number of parameter pairs may be attached to a header field value, any given parameter-name MUST NOT appear more than once.

When comparing header fields, field names are always case-insensitive. Unless otherwise stated in the definition of a particular header field, field values, parameter names, and parameter values are case-insensitive. Tokens are always case-insensitive. Unless specified otherwise, values expressed as quoted strings are case-sensitive. For example,

```
Contact: <sip:alice@atlanta.com>;expires=3600
```

is equivalent to

```
CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600
```

and

```
Content-Disposition: session;handling=optional
```

is equivalent to

```
content-disposition: Session;HANDLING=OPTIONAL
```

The following two header fields are not equivalent:

```
Warning: 370 devnull "Choose a bigger pipe"
Warning: 370 devnull "CHOOSE A BIGGER PIPE"
```

7.3.2 Header Field Classification

Some header fields only make sense in requests or responses. These are called request header fields and response header fields, respectively. If a header field appears in a message not matching its category (such as a request header field in a response), it MUST be ignored. Section 20 defines the classification of each header field.

7.3.3 Compact Form

SIP provides a mechanism to represent common header field names in an abbreviated form. This may be useful when messages would otherwise become too large to be carried on the transport available to it (exceeding the maximum transmission unit (MTU) when using UDP, for example). These compact forms are defined in Section 20. A compact form MAY be substituted for the longer form of a header field name at any time without changing the semantics of the message. A header

field name MAY appear in both long and short forms within the same message. Implementations MUST accept both the long and short forms of each header name.

7.4 Bodies

Requests, including new requests defined in extensions to this specification, MAY contain message bodies unless otherwise noted. The interpretation of the body depends on the request method.

For response messages, the request method and the response status code determine the type and interpretation of any message body. All responses MAY include a body.

7.4.1 Message Body Type

The Internet media type of the message body MUST be given by the Content-Type header field. If the body has undergone any encoding such as compression, then this MUST be indicated by the Content-Encoding header field; otherwise, Content-Encoding MUST be omitted. If applicable, the character set of the message body is indicated as part of the Content-Type header-field value.

The "multipart" MIME type defined in RFC 2046 [11] MAY be used within the body of the message. Implementations that send requests containing multipart message bodies MUST send a session description as a non-multipart message body if the remote implementation requests this through an Accept header field that does not contain multipart.

SIP messages MAY contain binary bodies or body parts. When no explicit charset parameter is provided by the sender, media subtypes of the "text" type are defined to have a default charset value of "UTF-8".

7.4.2 Message Body Length

The body length in bytes is provided by the Content-Length header field. Section 20.14 describes the necessary contents of this header field in detail.

The "chunked" transfer encoding of HTTP/1.1 MUST NOT be used for SIP. (Note: The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator.)

7.5 Framing SIP Messages

Unlike HTTP, SIP implementations can use UDP or other unreliable datagram protocols. Each such datagram carries one request or response. See Section 18 on constraints on usage of unreliable transports.

Implementations processing SIP messages over stream-oriented transports MUST ignore any CRLF appearing before the start-line [H4.1].

> The Content-Length header field value is used to locate the end of each SIP message in a stream. It will always be present when SIP messages are sent over stream-oriented transports.

8 General User Agent Behavior

A user agent represents an end system. It contains a user agent client (UAC), which generates requests, and a user agent server (UAS), which responds to them. A UAC is capable of generating a request based on some external stimulus (the user clicking a button, or a signal on a PSTN line) and processing a response. A UAS is capable of receiving a request and generating a response based on user input, external stimulus, the result of a program execution, or some other mechanism.

When a UAC sends a request, the request passes through some number of proxy servers, which forward the request towards the UAS. When the UAS generates a response, the response is forwarded towards the UAC.

UAC and UAS procedures depend strongly on two factors. First, based on whether the request or response is inside or outside of a dialog, and second, based on the method of a request. Dialogs are discussed thoroughly in Section 12; they represent a peer-to-peer relationship between user agents and are established by specific SIP methods, such as INVITE.

In this section, we discuss the method-independent rules for UAC and UAS behavior when processing requests that are outside of a dialog. This includes, of course, the requests which themselves establish a dialog.

Security procedures for requests and responses outside of a dialog are described in Section 26. Specifically, mechanisms exist for the UAS and UAC to mutually authenticate. A limited set of privacy features are also supported through encryption of bodies using S/MIME.

8.1 UAC Behavior

This section covers UAC behavior outside of a dialog.

8.1.1 Generating the Request

A valid SIP request formulated by a UAC MUST, at a minimum, contain the following header fields: To, From, CSeq, Call-ID, Max-Forwards, and Via; all of these header fields are mandatory in all SIP requests. These six header fields are the fundamental building blocks of a SIP message, as they jointly provide for most of the critical message routing services including the addressing of messages, the routing of responses, limiting message propagation, ordering of messages, and the unique identification of transactions. These header fields are in addition to the mandatory request line, which contains the method, Request-URI, and SIP version.

Examples of requests sent outside of a dialog include an INVITE to establish a session (Section 13) and an OPTIONS to query for capabilities (Section 11).

8.1.1.1 Request-URI

The initial Request-URI of the message SHOULD be set to the value of the URI in the To field. One notable exception is the REGISTER method; behavior for setting the Request-URI of REGISTER is given in Section 10. It may also be undesirable for privacy reasons or convenience to set these fields to the same value (especially if the originating UA expects that the Request-URI will be changed during transit).

In some special circumstances, the presence of a pre-existing route set can affect the Request-URI of the message. A pre-existing route set is an ordered set of URIs that identify a chain of servers, to which a UAC will send outgoing requests that are outside of a dialog. Commonly, they are configured on the UA by a user or service provider manually, or through some other non-SIP mechanism. When a provider wishes to configure a UA with an outbound proxy, it is RECOMMENDED that this be done by providing it with a pre-existing route set with a single URI, that of the outbound proxy.

When a pre-existing route set is present, the procedures for populating the Request-URI and Route header field detailed in Section 12.2.1.1 MUST be followed (even though there is no dialog), using the desired Request-URI as the remote target URI.

8.1.1.2 To

The To header field first and foremost specifies the desired "logical" recipient of the request, or the address-of-record of the user or resource that is the target of this request. This may or may not be the ultimate recipient of the request. The To header field MAY contain a SIP or SIPS URI, but it may also make use of other URI schemes (the tel URL (RFC 2806 [9]), for example) when appropriate. All SIP implementations MUST support the SIP URI scheme. Any implementation that supports TLS MUST support the SIPS URI scheme. The To header field allows for a display name.

A UAC may learn how to populate the To header field for a particular request in a number of ways. Usually the user will suggest the To header field through a human interface, perhaps inputting the URI manually or selecting it from some sort of address book. Frequently, the user will not enter a complete URI, but rather a string of digits or letters (for example, "bob"). It is at the discretion of the UA to choose how to interpret this input. Using the string to form the user part of a SIP URI implies that the UA wishes the name to be resolved in the domain to the right-hand side (RHS) of the at-sign in the SIP URI (for instance, sip:bob@example.com). Using the string to form the user part of a SIPS URI implies that the UA wishes to communicate securely, and that the name is to be resolved in the domain to the RHS of the at-sign. The RHS will frequently be the home domain of the requestor, which allows for the home domain to process the outgoing request. This is useful for features like "speed dial" that require interpretation of the user part in the home domain. The tel URL may be used when the UA does not wish to specify the domain that should interpret a telephone number that has been input by the user. Rather, each domain through which the request passes would be given that opportunity. As an example, a user in an airport might log in and send requests through an outbound proxy in the airport. If they enter "411" (this is the phone number for local directory assistance in the United States), that needs to be interpreted and processed by the outbound proxy in the airport, not the user's home domain. In this case, tel:411 would be the right choice.

A request outside of a dialog MUST NOT contain a To tag; the tag in the To field of a request identifies the peer of the dialog. Since no dialog is established, no tag is present.

For further information on the To header field, see Section 20.39. The following is an example of a valid To header field:

```
To: Carol <sip:carol@chicago.com>
```

8.1.1.3 From

The From header field indicates the logical identity of the initiator of the request, possibly the user's address-of-record. Like the To header field, it contains a URI and optionally a display name. It is used by SIP elements to determine which processing rules to apply to a request (for example, automatic call rejection). As such, it is very important that the From URI not contain IP addresses or the FQDN of the host on which the UA is running, since these are not logical names.

The From header field allows for a display name. A UAC SHOULD use the display name "Anonymous", along with a syntactically correct, but otherwise meaningless URI (like sip:thisis@anonymous.invalid), if the identity of the client is to remain hidden.

Usually, the value that populates the From header field in requests generated by a particular UA is pre-provisioned by the user or by the administrators of the user's local domain. If a particular UA is used by multiple users, it might have switchable profiles that include a URI corresponding to the identity of the profiled user. Recipients of requests can authenticate the originator of a request in order to ascertain that they are who their From header field claims they are (see Section 22 for more on authentication).

The From field MUST contain a new "tag" parameter, chosen by the UAC. See Section 19.3 for details on choosing a tag.

For further information on the From header field, see Section 20.20. Examples:

```
From: "Bob" <sips:bob@biloxi.com> ;tag=a48s
From: sip:+12125551212@phone2net.com;tag=887s
From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

8.1.1.4 Call-ID

The Call-ID header field acts as a unique identifier to group together a series of messages. It MUST be the same for all requests and responses sent by either UA in a dialog. It SHOULD be the same in each registration from a UA.

In a new request created by a UAC outside of any dialog, the Call-ID header field MUST be selected by the UAC as a globally unique identifier over space and time unless overridden by method-specific behavior. All SIP UAs must have a means to guarantee that the Call-ID header fields they produce will not be inadvertently generated by any other UA. Note that when requests are retried after certain

failure responses that solicit an amendment to a request (for example, a challenge for authentication), these retried requests are not considered new requests, and therefore do not need new Call-ID header fields; see Section 8.1.3.5.

Use of cryptographically random identifiers (RFC 1750 [12]) in the generation of Call-IDs is RECOMMENDED. Implementations MAY use the form "localid@host". Call-IDs are case-sensitive and are simply compared byte-by-byte.

> Using cryptographically random identifiers provides some protection against session hijacking and reduces the likelihood of unintentional Call-ID collisions.

No provisioning or human interface is required for the selection of the Call-ID header field value for a request.

For further information on the Call-ID header field, see Section 20.8.

Example:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com
```

8.1.1.5 CSeq

The CSeq header field serves as a way to identify and order transactions. It consists of a sequence number and a method. The method MUST match that of the request. For non-REGISTER requests outside of a dialog, the sequence number value is arbitrary. The sequence number value MUST be expressible as a 32-bit unsigned integer and MUST be less than 2**31. As long as it follows the above guidelines, a client may use any mechanism it would like to select CSeq header field values.

Section 12.2.1.1 discusses construction of the CSeq for requests within a dialog.

Example:

```
CSeq: 4711 INVITE
```

8.1.1.6 Max-Forwards

The Max-Forwards header field serves to limit the number of hops a request can transit on the way to its destination. It consists of an integer that is decremented by one at each hop. If the Max-Forwards value reaches 0 before the request reaches its destination, it will be rejected with a 483(Too Many Hops) error response.

A UAC MUST insert a Max-Forwards header field into each request it originates with a value that SHOULD be 70. This number was chosen to be sufficiently large to guarantee that a request would not be dropped in any SIP network when there were no loops, but not so large as to consume proxy resources when a loop does occur. Lower values should be used with caution and only in networks where topologies are known by the UA.

8.1.1.7 Via

The Via header field indicates the transport used for the transaction and identifies the location where the response is to be sent. A Via header field value is added only after the transport that will be used to reach the next hop has been selected (which may involve the usage of the procedures in [4]).

When the UAC creates a request, it MUST insert a Via into that request. The protocol name and protocol version in the header field MUST be SIP and 2.0, respectively. The Via header field value MUST contain a branch parameter. This parameter is used to identify the transaction created by that request. This parameter is used by both the client and the server.

The branch parameter value MUST be unique across space and time for all requests sent by the UA. The exceptions to this rule are CANCEL and ACK for non-2xx responses. As discussed below, a CANCEL request will have the same value of the branch parameter as the request it cancels. As discussed in Section 17.1.1.3, an ACK for a non-2xx response will also have the same branch ID as the INVITE whose response it acknowledges.

> The uniqueness property of the branch ID parameter, to facilitate its use as a transaction ID, was not part of RFC 2543.

The branch ID inserted by an element compliant with this specification MUST always begin with the characters "z9hG4bK". These 7 characters are used as a magic cookie (7 is deemed sufficient to ensure that an older RFC 2543 implementation would not pick such a value), so that servers receiving the request can determine that the branch ID was constructed in the fashion described by this

specification (that is, globally unique). Beyond this requirement, the precise format of the branch token is implementation-defined.

The Via header maddr, ttl, and sent-by components will be set when the request is processed by the transport layer (Section 18).

Via processing for proxies is described in Section 16.6 Item 8 and Section 16.7 Item 3.

8.1.1.8 Contact

The Contact header field provides a SIP or SIPS URI that can be used to contact that specific instance of the UA for subsequent requests. The Contact header field MUST be present and contain exactly one SIP or SIPS URI in any request that can result in the establishment of a dialog. For the methods defined in this specification, that includes only the INVITE request. For these requests, the scope of the Contact is global. That is, the Contact header field value contains the URI at which the UA would like to receive requests, and this URI MUST be valid even if used in subsequent requests outside of any dialogs.

If the Request-URI or top Route header field value contains a SIPS URI, the Contact header field MUST contain a SIPS URI as well.

For further information on the Contact header field, see Section 20.10.

8.1.1.9 Supported and Require

If the UAC supports extensions to SIP that can be applied by the server to the response, the UAC SHOULD include a Supported header field in the request listing the option tags (Section 19.2) for those extensions.

The option tags listed MUST only refer to extensions defined in standards-track RFCs. This is to prevent servers from insisting that clients implement non-standard, vendor-defined features in order to receive service. Extensions defined by experimental and informational RFCs are explicitly excluded from usage with the Supported header field in a request, since they too are often used to document vendor-defined extensions.

If the UAC wishes to insist that a UAS understand an extension that the UAC will apply to the request in order to process the request, it MUST insert a Require header field into the request listing the option tag for that extension. If the UAC wishes to apply an extension to the request and insist that any proxies that are

traversed understand that extension, it MUST insert a Proxy-Require header field into the request listing the option tag for that extension.

As with the Supported header field, the option tags in the Require and Proxy-Require header fields MUST only refer to extensions defined in standards-track RFCs.

8.1.1.10 Additional Message Components

After a new request has been created, and the header fields described above have been properly constructed, any additional optional header fields are added, as are any header fields specific to the method.

SIP requests MAY contain a MIME-encoded message-body. Regardless of the type of body that a request contains, certain header fields must be formulated to characterize the contents of the body. For further information on these header fields, see Sections 20.11 through 20.15.

8.1.2 Sending the Request

The destination for the request is then computed. Unless there is local policy specifying otherwise, the destination MUST be determined by applying the DNS procedures described in [4] as follows. If the first element in the route set indicated a strict router (resulting in forming the request as described in Section 12.2.1.1), the procedures MUST be applied to the Request-URI of the request. Otherwise, the procedures are applied to the first Route header field value in the request (if one exists), or to the request's Request-URI if there is no Route header field present. These procedures yield an ordered set of address, port, and transports to attempt. Independent of which URI is used as input to the procedures of [4], if the Request-URI specifies a SIPS resource, the UAC MUST follow the procedures of [4] as if the input URI were a SIPS URI.

Local policy MAY specify an alternate set of destinations to attempt. If the Request-URI contains a SIPS URI, any alternate destinations MUST be contacted with TLS. Beyond that, there are no restrictions on the alternate destinations if the request contains no Route header field. This provides a simple alternative to a pre-existing route set as a way to specify an outbound proxy. However, that approach for configuring an outbound proxy is NOT RECOMMENDED; a pre-existing route set with a single URI SHOULD be used instead. If the request contains a Route header field, the request SHOULD be sent to the locations derived from its topmost value, but MAY be sent to any server that the UA is certain will honor the Route and Request-URI policies specified in this document (as opposed to those in RFC 2543). In particular, a UAC configured with an outbound proxy SHOULD

attempt to send the request to the location indicated in the first Route header field value instead of adopting the policy of sending all messages to the outbound proxy.

> This ensures that outbound proxies that do not add Record-Route header field values will drop out of the path of subsequent requests. It allows endpoints that cannot resolve the first Route URI to delegate that task to an outbound proxy.

The UAC SHOULD follow the procedures defined in [4] for stateful elements, trying each address until a server is contacted. Each try constitutes a new transaction, and therefore each carries a different topmost Via header field value with a new branch parameter. Furthermore, the transport value in the Via header field is set to whatever transport was determined for the target server.

8.1.3 Processing Responses

Responses are first processed by the transport layer and then passed up to the transaction layer. The transaction layer performs its processing and then passes the response up to the TU. The majority of response processing in the TU is method specific. However, there are some general behaviors independent of the method.

8.1.3.1 Transaction Layer Errors

In some cases, the response returned by the transaction layer will not be a SIP message, but rather a transaction layer error. When a timeout error is received from the transaction layer, it MUST be treated as if a 408 (Request Timeout) status code has been received. If a fatal transport error is reported by the transport layer (generally, due to fatal ICMP errors in UDP or connection failures in TCP), the condition MUST be treated as a 503 (Service Unavailable) status code.

8.1.3.2 Unrecognized Responses

A UAC MUST treat any final response it does not recognize as being equivalent to the x00 response code of that class, and MUST be able to process the x00 response code for all classes. For example, if a UAC receives an unrecognized response code of 431, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 (Bad Request) response code. A UAC MUST treat any provisional response different than 100 that it does not recognize as 183 (Session Progress). A UAC MUST be able to process 100 and 183 responses.

8.1.3.3 Vias

If more than one Via header field value is present in a response, the UAC SHOULD discard the message.

> The presence of additional Via header field values that precede the originator of the request suggests that the message was misrouted or possibly corrupted.

8.1.3.4 Processing 3xx Responses

Upon receipt of a redirection response (for example, a 301 response status code), clients SHOULD use the URI(s) in the Contact header field to formulate one or more new requests based on the redirected request. This process is similar to that of a proxy recursing on a 3xx class response as detailed in Sections 16.5 and 16.6. A client starts with an initial target set containing exactly one URI, the Request-URI of the original request. If a client wishes to formulate new requests based on a 3xx class response to that request, it places the URIs to try into the target set. Subject to the restrictions in this specification, a client can choose which Contact URIs it places into the target set. As with proxy recursion, a client processing 3xx class responses MUST NOT add any given URI to the target set more than once. If the original request had a SIPS URI in the Request-URI, the client MAY choose to recurse to a non-SIPS URI, but SHOULD inform the user of the redirection to an insecure URI.

> Any new request may receive 3xx responses themselves containing the original URI as a contact. Two locations can be configured to redirect to each other. Placing any given URI in the target set only once prevents infinite redirection loops.

As the target set grows, the client MAY generate new requests to the URIs in any order. A common mechanism is to order the set by the "q" parameter value from the Contact header field value. Requests to the URIs MAY be generated serially or in parallel. One approach is to process groups of decreasing q-values serially and process the URIs in each q-value group in parallel. Another is to perform only serial processing in decreasing q-value order, arbitrarily choosing between contacts of equal q-value.

If contacting an address in the list results in a failure, as defined in the next paragraph, the element moves to the next address in the list, until the list is exhausted. If the list is exhausted, then the request has failed.

Failures SHOULD be detected through failure response codes (codes greater than 399); for network errors the client transaction will report any transport layer failures to the transaction user. Note that some response codes (detailed in 8.1.3.5) indicate that the request can be retried; requests that are reattempted should not be considered failures.

When a failure for a particular contact address is received, the client SHOULD try the next contact address. This will involve creating a new client transaction to deliver a new request.

In order to create a request based on a contact address in a 3xx response, a UAC MUST copy the entire URI from the target set into the Request-URI, except for the "method-param" and "header" URI parameters (see Section 19.1.1 for a definition of these parameters). It uses the "header" parameters to create header field values for the new request, overwriting header field values associated with the redirected request in accordance with the guidelines in Section 19.1.5.

Note that in some instances, header fields that have been communicated in the contact address may instead append to existing request header fields in the original redirected request. As a general rule, if the header field can accept a comma-separated list of values, then the new header field value MAY be appended to any existing values in the original redirected request. If the header field does not accept multiple values, the value in the original redirected request MAY be overwritten by the header field value communicated in the contact address. For example, if a contact address is returned with the following value:

```
sip:user@host?Subject=foo&Call-Info=<http://www.foo.com>
```

Then any Subject header field in the original redirected request is overwritten, but the HTTP URL is merely appended to any existing Call-Info header field values.

It is RECOMMENDED that the UAC reuse the same To, From, and Call-ID used in the original redirected request, but the UAC MAY also choose to update the Call-ID header field value for new requests, for example.

Finally, once the new request has been constructed, it is sent using a new client transaction, and therefore MUST have a new branch ID in the top Via field as discussed in Section 8.1.1.7.

In all other respects, requests sent upon receipt of a redirect response SHOULD re-use the header fields and bodies of the original request.

In some instances, Contact header field values may be cached at UAC temporarily or permanently depending on the status code received and the presence of an expiration interval; see Sections 21.3.2 and 21.3.3.

8.1.3.5 Processing 4xx Responses

Certain 4xx response codes require specific UA processing, independent of the method.

If a 401 (Unauthorized) or 407 (Proxy Authentication Required) response is received, the UAC SHOULD follow the authorization procedures of Section 22.2 and Section 22.3 to retry the request with credentials.

If a 413 (Request Entity Too Large) response is received (Section 21.4.11), the request contained a body that was longer than the UAS was willing to accept. If possible, the UAC SHOULD retry the request, either omitting the body or using one of a smaller length.

If a 415 (Unsupported Media Type) response is received (Section 21.4.13), the request contained media types not supported by the UAS. The UAC SHOULD retry sending the request, this time only using content with types listed in the Accept header field in the response, with encodings listed in the Accept-Encoding header field in the response, and with languages listed in the Accept-Language in the response.

If a 416 (Unsupported URI Scheme) response is received (Section 21.4.14), the Request-URI used a URI scheme not supported by the server. The client SHOULD retry the request, this time, using a SIP URI.

If a 420 (Bad Extension) response is received (Section 21.4.15), the request contained a Require or Proxy-Require header field listing an option-tag for a feature not supported by a proxy or UAS. The UAC SHOULD retry the request, this time omitting any extensions listed in the Unsupported header field in the response.

In all of the above cases, the request is retried by creating a new request with the appropriate modifications. This new request constitutes a new transaction and SHOULD have the same value of the Call-ID, To, and From of the previous request, but the CSeq should contain a new sequence number that is one higher than the previous.

With other 4xx responses, including those yet to be defined, a retry may or may not be possible depending on the method and the use case.

8.2 UAS Behavior

When a request outside of a dialog is processed by a UAS, there is a set of processing rules that are followed, independent of the method. Section 12 gives guidance on how a UAS can tell whether a request is inside or outside of a dialog.

Note that request processing is atomic. If a request is accepted, all state changes associated with it MUST be performed. If it is rejected, all state changes MUST NOT be performed.

UASs SHOULD process the requests in the order of the steps that follow in this section (that is, starting with authentication, then inspecting the method, the header fields, and so on throughout the remainder of this section).

8.2.1 Method Inspection

Once a request is authenticated (or authentication is skipped), the UAS MUST inspect the method of the request. If the UAS recognizes but does not support the method of a request, it MUST generate a 405 (Method Not Allowed) response. Procedures for generating responses are described in Section 8.2.6. The UAS MUST also add an Allow header field to the 405 (Method Not Allowed) response. The Allow header field MUST list the set of methods supported by the UAS generating the message. The Allow header field is presented in Section 20.5.

If the method is one supported by the server, processing continues.

8.2.2 Header Inspection

If a UAS does not understand a header field in a request (that is, the header field is not defined in this specification or in any supported extension), the server MUST ignore that header field and continue processing the message. A UAS SHOULD ignore any malformed header fields that are not necessary for processing requests.

8.2.2.1 To and Request-URI

The To header field identifies the original recipient of the request designated by the user identified in the From field. The original recipient may or may not be the UAS processing the request, due to call forwarding or other proxy operations. A UAS MAY apply any policy it wishes to determine whether to accept requests when the To

header field is not the identity of the UAS. However, it is RECOMMENDED that a UAS accept requests even if they do not recognize the URI scheme (for example, a tel: URI) in the To header field, or if the To header field does not address a known or current user of this UAS. If, on the other hand, the UAS decides to reject the request, it SHOULD generate a response with a 403 (Forbidden) status code and pass it to the server transaction for transmission.

However, the Request-URI identifies the UAS that is to process the request. If the Request-URI uses a scheme not supported by the UAS, it SHOULD reject the request with a 416 (Unsupported URI Scheme) response. If the Request-URI does not identify an address that the UAS is willing to accept requests for, it SHOULD reject the request with a 404 (Not Found) response. Typically, a UA that uses the REGISTER method to bind its address-of-record to a specific contact address will see requests whose Request-URI equals that contact address. Other potential sources of received Request-URIs include the Contact header fields of requests and responses sent by the UA that establish or refresh dialogs.

8.2.2.2 Merged Requests

If the request has no tag in the To header field, the UAS core MUST check the request against ongoing transactions. If the From tag, Call-ID, and CSeq exactly match those associated with an ongoing transaction, but the request does not match that transaction (based on the matching rules in Section 17.2.3), the UAS core SHOULD generate a 482 (Loop Detected) response and pass it to the server transaction.

> The same request has arrived at the UAS more than once, following different paths, most likely due to forking. The UAS processes the first such request received and responds with a 482 (Loop Detected) to the rest of them.

8.2.2.3 Require

Assuming the UAS decides that it is the proper element to process the request, it examines the Require header field, if present.

The Require header field is used by a UAC to tell a UAS about SIP extensions that the UAC expects the UAS to support in order to process the request properly. Its format is described in Section 20.32. If a UAS does not understand an option-tag listed in a Require header field, it MUST respond by generating a response with status code 420 (Bad Extension). The UAS MUST add an Unsupported header field, and list in it those options it does not understand amongst those in the Require header field of the request.

Note that Require and Proxy-Require MUST NOT be used in a SIP CANCEL request, or in an ACK request sent for a non-2xx response. These header fields MUST be ignored if they are present in these requests.

An ACK request for a 2xx response MUST contain only those Require and Proxy-Require values that were present in the initial request.

Example:

```
UAC->UAS:   INVITE sip:watson@bell-telephone.com SIP/2.0
            Require: 100rel

UAS->UAC:   SIP/2.0 420 Bad Extension
            Unsupported: 100rel
```

This behavior ensures that the client-server interaction will proceed without delay when all options are understood by both sides, and only slow down if options are not understood (as in the example above). For a well-matched client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms. In addition, it also removes ambiguity when the client requires features that the server does not understand. Some features, such as call handling fields, are only of interest to end systems.

8.2.3 Content Processing

Assuming the UAS understands any extensions required by the client, the UAS examines the body of the message, and the header fields that describe it. If there are any bodies whose type (indicated by the Content-Type), language (indicated by the Content-Language) or encoding (indicated by the Content-Encoding) are not understood, and that body part is not optional (as indicated by the Content-Disposition header field), the UAS MUST reject the request with a 415 (Unsupported Media Type) response. The response MUST contain an Accept header field listing the types of all bodies it understands, in the event the request contained bodies of types not supported by the UAS. If the request contained content encodings not understood by the UAS, the response MUST contain an Accept-Encoding header field listing the encodings understood by the UAS. If the request contained content with languages not understood by the UAS, the response MUST contain an Accept-Language header field indicating the languages understood by the UAS. Beyond these checks, body handling depends on the method and type. For further information on the processing of content-specific header fields, see Section 7.4 as well as Section 20.11 through 20.15.

8.2.4 Applying Extensions

A UAS that wishes to apply some extension when generating the response MUST NOT do so unless support for that extension is indicated in the Supported header field in the request. If the desired extension is not supported, the server SHOULD rely only on baseline SIP and any other extensions supported by the client. In rare circumstances, where the server cannot process the request without the extension, the server MAY send a 421 (Extension Required) response. This response indicates that the proper response cannot be generated without support of a specific extension. The needed extension(s) MUST be included in a Require header field in the response. This behavior is NOT RECOMMENDED, as it will generally break interoperability.

Any extensions applied to a non-421 response MUST be listed in a Require header field included in the response. Of course, the server MUST NOT apply extensions not listed in the Supported header field in the request. As a result of this, the Require header field in a response will only ever contain option tags defined in standards-track RFCs.

8.2.5 Processing the Request

Assuming all of the checks in the previous subsections are passed, the UAS processing becomes method-specific. Section 10 covers the REGISTER request, Section 11 covers the OPTIONS request, Section 13 covers the INVITE request, and Section 15 covers the BYE request.

8.2.6 Generating the Response

When a UAS wishes to construct a response to a request, it follows the general procedures detailed in the following subsections. Additional behaviors specific to the response code in question, which are not detailed in this section, may also be required.

Once all procedures associated with the creation of a response have been completed, the UAS hands the response back to the server transaction from which it received the request.

8.2.6.1 Sending a Provisional Response

One largely non-method-specific guideline for the generation of responses is that UASs SHOULD NOT issue a provisional response for a non-INVITE request. Rather, UASs SHOULD generate a final response to a non-INVITE request as soon as possible.

When a 100 (Trying) response is generated, any Timestamp header field present in the request MUST be copied into this 100 (Trying) response. If there is a delay in generating the response, the UAS SHOULD add a delay value into the Timestamp value in the response. This value MUST contain the difference between the time of sending of the response and receipt of the request, measured in seconds.

8.2.6.2 Headers and Tags

The From field of the response MUST equal the From header field of the request. The Call-ID header field of the response MUST equal the Call-ID header field of the request. The CSeq header field of the response MUST equal the CSeq field of the request. The Via header field values in the response MUST equal the Via header field values in the request and MUST maintain the same ordering.

If a request contained a To tag in the request, the To header field in the response MUST equal that of the request. However, if the To header field in the request did not contain a tag, the URI in the To header field in the response MUST equal the URI in the To header field; additionally, the UAS MUST add a tag to the To header field in the response (with the exception of the 100 (Trying) response, in which a tag MAY be present). This serves to identify the UAS that is responding, possibly resulting in a component of a dialog ID. The same tag MUST be used for all responses to that request, both final and provisional (again excepting the 100 (Trying)). Procedures for the generation of tags are defined in Section 19.3.

8.2.7 Stateless UAS Behavior

A stateless UAS is a UAS that does not maintain transaction state. It replies to requests normally, but discards any state that would ordinarily be retained by a UAS after a response has been sent. If a stateless UAS receives a retransmission of a request, it regenerates the response and resends it, just as if it were replying to the first instance of the request. A UAS cannot be stateless unless the request processing for that method would always result in the same response if the requests are identical. This rules out stateless registrars, for example. Stateless UASs do not use a transaction layer; they receive requests directly from the transport layer and send responses directly to the transport layer.

The stateless UAS role is needed primarily to handle unauthenticated requests for which a challenge response is issued. If unauthenticated requests were handled statefully, then malicious floods of unauthenticated requests could create massive amounts of

transaction state that might slow or completely halt call processing in a UAS, effectively creating a denial of service condition; for more information see Section 26.1.5.

The most important behaviors of a stateless UAS are the following:

- A stateless UAS MUST NOT send provisional (1xx) responses.
- A stateless UAS MUST NOT retransmit responses.
- A stateless UAS MUST ignore ACK requests.
- A stateless UAS MUST ignore CANCEL requests.
- To header tags MUST be generated for responses in a stateless manner - in a manner that will generate the same tag for the same request consistently. For information on tag construction see Section 19.3.

In all other respects, a stateless UAS behaves in the same manner as a stateful UAS. A UAS can operate in either a stateful or stateless mode for each new request.

8.3 Redirect Servers

In some architectures it may be desirable to reduce the processing load on proxy servers that are responsible for routing requests, and improve signaling path robustness, by relying on redirection.

Redirection allows servers to push routing information for a request back in a response to the client, thereby taking themselves out of the loop of further messaging for this transaction while still aiding in locating the target of the request. When the originator of the request receives the redirection, it will send a new request based on the URI(s) it has received. By propagating URIs from the core of the network to its edges, redirection allows for considerable network scalability.

A redirect server is logically constituted of a server transaction layer and a transaction user that has access to a location service of some kind (see Section 10 for more on registrars and location services). This location service is effectively a database containing mappings between a single URI and a set of one or more alternative locations at which the target of that URI can be found.

A redirect server does not issue any SIP requests of its own. After receiving a request other than CANCEL, the server either refuses the request or gathers the list of alternative locations from the

location service and returns a final response of class 3xx. For well-formed CANCEL requests, it SHOULD return a 2xx response. This response ends the SIP transaction. The redirect server maintains transaction state for an entire SIP transaction. It is the responsibility of clients to detect forwarding loops between redirect servers.

When a redirect server returns a 3xx response to a request, it populates the list of (one or more) alternative locations into the Contact header field. An "expires" parameter to the Contact header field values may also be supplied to indicate the lifetime of the Contact data.

The Contact header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 301 (Moved Permanently) or 302 (Moved Temporarily) response may also give the same location and username that was targeted by the initial request but specify additional transport parameters such as a different server or multicast address to try, or a change of SIP transport from UDP to TCP or vice versa.

However, redirect servers MUST NOT redirect a request to a URI equal to the one in the Request-URI; instead, provided that the URI does not point to itself, the server MAY proxy the request to the destination URI, or MAY reject it with a 404.

> If a client is using an outbound proxy, and that proxy actually redirects requests, a potential arises for infinite redirection loops.

Note that a Contact header field value MAY also refer to a different resource than the one originally called. For example, a SIP call connected to PSTN gateway may need to deliver a special informational announcement such as "The number you have dialed has been changed."

A Contact response header field can contain any suitable URI indicating where the called party can be reached, not limited to SIP URIs. For example, it could contain URIs for phones, fax, or irc (if they were defined) or a mailto: (RFC 2368 [32]) URL. Section 26.4.4 discusses implications and limitations of redirecting a SIPS URI to a non-SIPS URI.

The "expires" parameter of a Contact header field value indicates how long the URI is valid. The value of the parameter is a number indicating seconds. If this parameter is not provided, the value of the Expires header field determines how long the URI is valid. Malformed values SHOULD be treated as equivalent to 3600.

This provides a modest level of backwards compatibility with RFC 2543, which allowed absolute times in this header field. If an absolute time is received, it will be treated as malformed, and then default to 3600.

Redirect servers MUST ignore features that are not understood (including unrecognized header fields, any unknown option tags in Require, or even method names) and proceed with the redirection of the request in question.

9 Canceling a Request

The previous section has discussed general UA behavior for generating requests and processing responses for requests of all methods. In this section, we discuss a general purpose method, called CANCEL.

The CANCEL request, as the name implies, is used to cancel a previous request sent by a client. Specifically, it asks the UAS to cease processing the request and to generate an error response to that request. CANCEL has no effect on a request to which a UAS has already given a final response. Because of this, it is most useful to CANCEL requests to which it can take a server long time to respond. For this reason, CANCEL is best for INVITE requests, which can take a long time to generate a response. In that usage, a UAS that receives a CANCEL request for an INVITE, but has not yet sent a final response, would "stop ringing", and then respond to the INVITE with a specific error response (a 487).

CANCEL requests can be constructed and sent by both proxies and user agent clients. Section 15 discusses under what conditions a UAC would CANCEL an INVITE request, and Section 16.10 discusses proxy usage of CANCEL.

A stateful proxy responds to a CANCEL, rather than simply forwarding a response it would receive from a downstream element. For that reason, CANCEL is referred to as a "hop-by-hop" request, since it is responded to at each stateful proxy hop.

9.1 Client Behavior

A CANCEL request SHOULD NOT be sent to cancel a request other than INVITE.

Since requests other than INVITE are responded to immediately, sending a CANCEL for a non-INVITE request would always create a race condition.

The following procedures are used to construct a CANCEL request. The Request-URI, Call-ID, To, the numeric part of CSeq, and From header fields in the CANCEL request MUST be identical to those in the request being cancelled, including tags. A CANCEL constructed by a client MUST have only a single Via header field value matching the top Via value in the request being cancelled. Using the same values for these header fields allows the CANCEL to be matched with the request it cancels (Section 9.2 indicates how such matching occurs). However, the method part of the CSeq header field MUST have a value of CANCEL. This allows it to be identified and processed as a transaction in its own right (See Section 17).

If the request being cancelled contains a Route header field, the CANCEL request MUST include that Route header field's values.

> This is needed so that stateless proxies are able to route CANCEL requests properly.

The CANCEL request MUST NOT contain any Require or Proxy-Require header fields.

Once the CANCEL is constructed, the client SHOULD check whether it has received any response (provisional or final) for the request being cancelled (herein referred to as the "original request").

If no provisional response has been received, the CANCEL request MUST NOT be sent; rather, the client MUST wait for the arrival of a provisional response before sending the request. If the original request has generated a final response, the CANCEL SHOULD NOT be sent, as it is an effective no-op, since CANCEL has no effect on requests that have already generated a final response. When the client decides to send the CANCEL, it creates a client transaction for the CANCEL and passes it the CANCEL request along with the destination address, port, and transport. The destination address, port, and transport for the CANCEL MUST be identical to those used to send the original request.

> If it was allowed to send the CANCEL before receiving a response for the previous request, the server could receive the CANCEL before the original request.

Note that both the transaction corresponding to the original request and the CANCEL transaction will complete independently. However, a UAC canceling a request cannot rely on receiving a 487 (Request Terminated) response for the original request, as an RFC 2543-compliant UAS will not generate such a response. If there is no final response for the original request in 64*T1 seconds (T1 is

defined in Section 17.1.1.1), the client SHOULD then consider the original transaction cancelled and SHOULD destroy the client transaction handling the original request.

9.2 Server Behavior

The CANCEL method requests that the TU at the server side cancel a pending transaction. The TU determines the transaction to be cancelled by taking the CANCEL request, and then assuming that the request method is anything but CANCEL or ACK and applying the transaction matching procedures of Section 17.2.3. The matching transaction is the one to be cancelled.

The processing of a CANCEL request at a server depends on the type of server. A stateless proxy will forward it, a stateful proxy might respond to it and generate some CANCEL requests of its own, and a UAS will respond to it. See Section 16.10 for proxy treatment of CANCEL.

A UAS first processes the CANCEL request according to the general UAS processing described in Section 8.2. However, since CANCEL requests are hop-by-hop and cannot be resubmitted, they cannot be challenged by the server in order to get proper credentials in an Authorization header field. Note also that CANCEL requests do not contain a Require header field.

If the UAS did not find a matching transaction for the CANCEL according to the procedure above, it SHOULD respond to the CANCEL with a 481 (Call Leg/Transaction Does Not Exist). If the transaction for the original request still exists, the behavior of the UAS on receiving a CANCEL request depends on whether it has already sent a final response for the original request. If it has, the CANCEL request has no effect on the processing of the original request, no effect on any session state, and no effect on the responses generated for the original request. If the UAS has not issued a final response for the original request, its behavior depends on the method of the original request. If the original request was an INVITE, the UAS SHOULD immediately respond to the INVITE with a 487 (Request Terminated). A CANCEL request has no impact on the processing of transactions with any other method defined in this specification.

Regardless of the method of the original request, as long as the CANCEL matched an existing transaction, the UAS answers the CANCEL request itself with a 200 (OK) response. This response is constructed following the procedures described in Section 8.2.6 noting that the To tag of the response to the CANCEL and the To tag in the response to the original request SHOULD be the same. The response to CANCEL is passed to the server transaction for transmission.

10 Registrations

10.1 Overview

SIP offers a discovery capability. If a user wants to initiate a session with another user, SIP must discover the current host(s) at which the destination user is reachable. This discovery process is frequently accomplished by SIP network elements such as proxy servers and redirect servers which are responsible for receiving a request, determining where to send it based on knowledge of the location of the user, and then sending it there. To do this, SIP network elements consult an abstract service known as a location service, which provides address bindings for a particular domain. These address bindings map an incoming SIP or SIPS URI, sip:bob@biloxi.com, for example, to one or more URIs that are somehow "closer" to the desired user, sip:bob@engineering.biloxi.com, for example. Ultimately, a proxy will consult a location service that maps a received URI to the user agent(s) at which the desired recipient is currently residing.

Registration creates bindings in a location service for a particular domain that associates an address-of-record URI with one or more contact addresses. Thus, when a proxy for that domain receives a request whose Request-URI matches the address-of-record, the proxy will forward the request to the contact addresses registered to that address-of-record. Generally, it only makes sense to register an address-of-record at a domain's location service when requests for that address-of-record would be routed to that domain. In most cases, this means that the domain of the registration will need to match the domain in the URI of the address-of-record.

There are many ways by which the contents of the location service can be established. One way is administratively. In the above example, Bob is known to be a member of the engineering department through access to a corporate database. However, SIP provides a mechanism for a UA to create a binding explicitly. This mechanism is known as registration.

Registration entails sending a REGISTER request to a special type of UAS known as a registrar. A registrar acts as the front end to the location service for a domain, reading and writing mappings based on the contents of REGISTER requests. This location service is then typically consulted by a proxy server that is responsible for routing requests for that domain.

An illustration of the overall registration process is given in Figure 2. Note that the registrar and proxy server are logical roles that can be played by a single device in a network; for purposes of

clarity the two are separated in this illustration. Also note that UAs may send requests through a proxy server in order to reach a registrar if the two are separate elements.

SIP does not mandate a particular mechanism for implementing the location service. The only requirement is that a registrar for some domain MUST be able to read and write data to the location service, and a proxy or a redirect server for that domain MUST be capable of reading that same data. A registrar MAY be co-located with a particular SIP proxy server for the same domain.

10.2 Constructing the REGISTER Request

REGISTER requests add, remove, and query bindings. A REGISTER request can add a new binding between an address-of-record and one or more contact addresses. Registration on behalf of a particular address-of-record can be performed by a suitably authorized third party. A client can also remove previous bindings or query to determine which bindings are currently in place for an address-of-record.

Except as noted, the construction of the REGISTER request and the behavior of clients sending a REGISTER request is identical to the general UAC behavior described in Section 8.1 and Section 17.1.

A REGISTER request does not establish a dialog. A UAC MAY include a Route header field in a REGISTER request based on a pre-existing route set as described in Section 8.1. The Record-Route header field has no meaning in REGISTER requests or responses, and MUST be ignored if present. In particular, the UAC MUST NOT create a new route set based on the presence or absence of a Record-Route header field in any response to a REGISTER request.

The following header fields, except Contact, MUST be included in a REGISTER request. A Contact header field MAY be included:

Request-URI: The Request-URI names the domain of the location service for which the registration is meant (for example, "sip:chicago.com"). The "userinfo" and "@" components of the SIP URI MUST NOT be present.

To: The To header field contains the address of record whose registration is to be created, queried, or modified. The To header field and the Request-URI field typically differ, as the former contains a user name. This address-of-record MUST be a SIP URI or SIPS URI.

From: The From header field contains the address-of-record of the person responsible for the registration. The value is the same as the To header field unless the request is a third-party registration.

Call-ID: All registrations from a UAC SHOULD use the same Call-ID header field value for registrations sent to a particular registrar.

If the same client were to use different Call-ID values, a registrar could not detect whether a delayed REGISTER request might have arrived out of order.

CSeq: The CSeq value guarantees proper ordering of REGISTER requests. A UA MUST increment the CSeq value by one for each REGISTER request with the same Call-ID.

Contact: REGISTER requests MAY contain a Contact header field with zero or more values containing address bindings.

UAs MUST NOT send a new registration (that is, containing new Contact header field values, as opposed to a retransmission) until they have received a final response from the registrar for the previous one or the previous REGISTER request has timed out.

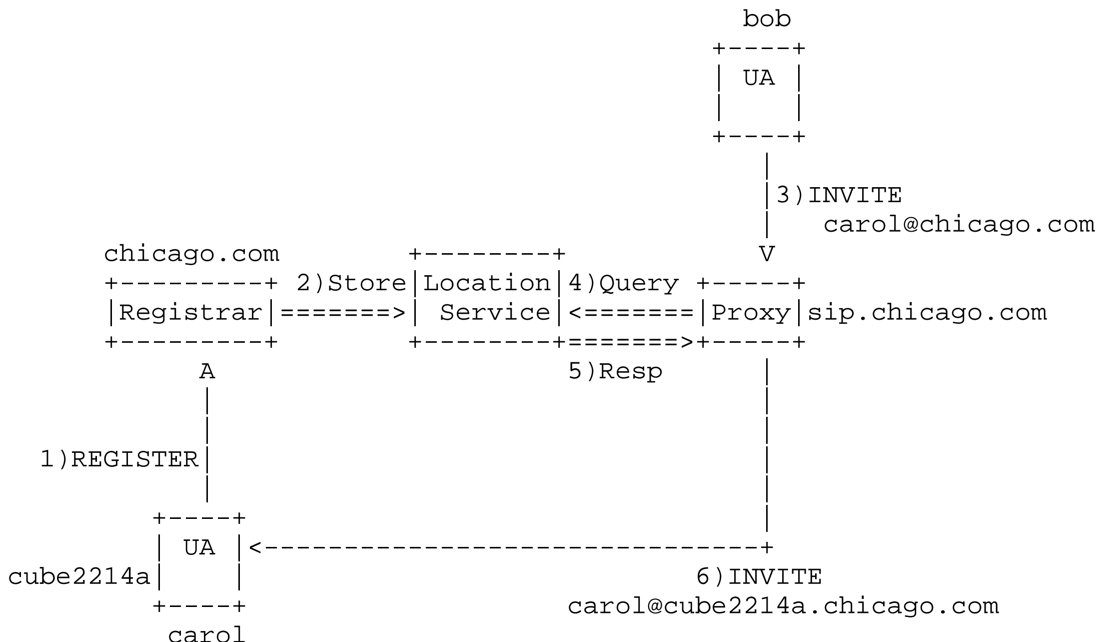


Figure 2: REGISTER example

The following Contact header parameters have a special meaning in REGISTER requests:

action: The "action" parameter from RFC 2543 has been deprecated. UACs SHOULD NOT use the "action" parameter.

expires: The "expires" parameter indicates how long the UA would like the binding to be valid. The value is a number indicating seconds. If this parameter is not provided, the value of the Expires header field is used instead. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as equivalent to 2**32-1. Malformed values SHOULD be treated as equivalent to 3600.

10.2.1 Adding Bindings

The REGISTER request sent to a registrar includes the contact address(es) to which SIP requests for the address-of-record should be forwarded. The address-of-record is included in the To header field of the REGISTER request.

The Contact header field values of the request typically consist of SIP or SIPS URIs that identify particular SIP endpoints (for example, "sip:carol@cube2214a.chicago.com"), but they MAY use any URI scheme. A SIP UA can choose to register telephone numbers (with the tel URL, RFC 2806 [9]) or email addresses (with a mailto URL, RFC 2368 [32]) as Contacts for an address-of-record, for example.

For example, Carol, with address-of-record "sip:carol@chicago.com", would register with the SIP registrar of the domain chicago.com. Her registrations would then be used by a proxy server in the chicago.com domain to route requests for Carol's address-of-record to her SIP endpoint.

Once a client has established bindings at a registrar, it MAY send subsequent registrations containing new bindings or modifications to existing bindings as necessary. The 2xx response to the REGISTER request will contain, in a Contact header field, a complete list of bindings that have been registered for this address-of-record at this registrar.

If the address-of-record in the To header field of a REGISTER request is a SIPS URI, then any Contact header field values in the request SHOULD also be SIPS URIs. Clients should only register non-SIPS URIs under a SIPS address-of-record when the security of the resource represented by the contact address is guaranteed by other means. This may be applicable to URIs that invoke protocols other than SIP, or SIP devices secured by protocols other than TLS.

Registrations do not need to update all bindings. Typically, a UA only updates its own contact addresses.

10.2.1.1 Setting the Expiration Interval of Contact Addresses

When a client sends a REGISTER request, it MAY suggest an expiration interval that indicates how long the client would like the registration to be valid. (As described in Section 10.3, the registrar selects the actual time interval based on its local policy.)

There are two ways in which a client can suggest an expiration interval for a binding: through an Expires header field or an "expires" Contact header parameter. The latter allows expiration intervals to be suggested on a per-binding basis when more than one binding is given in a single REGISTER request, whereas the former suggests an expiration interval for all Contact header field values that do not contain the "expires" parameter.

If neither mechanism for expressing a suggested expiration time is present in a REGISTER, the client is indicating its desire for the server to choose.

10.2.1.2 Preferences among Contact Addresses

If more than one Contact is sent in a REGISTER request, the registering UA intends to associate all of the URIs in these Contact header field values with the address-of-record present in the To field. This list can be prioritized with the "q" parameter in the Contact header field. The "q" parameter indicates a relative preference for the particular Contact header field value compared to other bindings for this address-of-record. Section 16.6 describes how a proxy server uses this preference indication.

10.2.2 Removing Bindings

Registrations are soft state and expire unless refreshed, but can also be explicitly removed. A client can attempt to influence the expiration interval selected by the registrar as described in Section 10.2.1. A UA requests the immediate removal of a binding by specifying an expiration interval of "0" for that contact address in a REGISTER request. UAs SHOULD support this mechanism so that bindings can be removed before their expiration interval has passed.

The REGISTER-specific Contact header field value of "*" applies to all registrations, but it MUST NOT be used unless the Expires header field is present with a value of "0".

> Use of the "*" Contact header field value allows a registering UA to remove all bindings associated with an address-of-record without knowing their precise values.

10.2.3 Fetching Bindings

A success response to any REGISTER request contains the complete list of existing bindings, regardless of whether the request contained a Contact header field. If no Contact header field is present in a REGISTER request, the list of bindings is left unchanged.

10.2.4 Refreshing Bindings

Each UA is responsible for refreshing the bindings that it has previously established. A UA SHOULD NOT refresh bindings set up by other UAs.

The 200 (OK) response from the registrar contains a list of Contact fields enumerating all current bindings. The UA compares each contact address to see if it created the contact address, using comparison rules in Section 19.1.4. If so, it updates the expiration time interval according to the expires parameter or, if absent, the Expires field value. The UA then issues a REGISTER request for each of its bindings before the expiration interval has elapsed. It MAY combine several updates into one REGISTER request.

A UA SHOULD use the same Call-ID for all registrations during a single boot cycle. Registration refreshes SHOULD be sent to the same network address as the original registration, unless redirected.

10.2.5 Setting the Internal Clock

If the response for a REGISTER request contains a Date header field, the client MAY use this header field to learn the current time in order to set any internal clocks.

10.2.6 Discovering a Registrar

UAs can use three ways to determine the address to which to send registrations: by configuration, using the address-of-record, and multicast. A UA can be configured, in ways beyond the scope of this specification, with a registrar address. If there is no configured registrar address, the UA SHOULD use the host part of the address-of-record as the Request-URI and address the request there, using the normal SIP server location mechanisms [4]. For example, the UA for the user "sip:carol@chicago.com" addresses the REGISTER request to "sip:chicago.com".

Finally, a UA can be configured to use multicast. Multicast registrations are addressed to the well-known "all SIP servers" multicast address "sip.mcast.net" (224.0.1.75 for IPv4). No well-known IPv6 multicast address has been allocated; such an allocation will be documented separately when needed. SIP UAs MAY listen to that address and use it to become aware of the location of other local users (see [33]); however, they do not respond to the request.

> Multicast registration may be inappropriate in some environments, for example, if multiple businesses share the same local area network.

10.2.7 Transmitting a Request

Once the REGISTER method has been constructed, and the destination of the message identified, UACs follow the procedures described in Section 8.1.2 to hand off the REGISTER to the transaction layer.

If the transaction layer returns a timeout error because the REGISTER yielded no response, the UAC SHOULD NOT immediately re-attempt a registration to the same registrar.

An immediate re-attempt is likely to also timeout. Waiting some reasonable time interval for the conditions causing the timeout to be corrected reduces unnecessary load on the network. No specific interval is mandated.

10.2.8 Error Responses

If a UA receives a 423 (Interval Too Brief) response, it MAY retry the registration after making the expiration interval of all contact addresses in the REGISTER request equal to or greater than the expiration interval within the Min-Expires header field of the 423 (Interval Too Brief) response.

10.3 Processing REGISTER Requests

A registrar is a UAS that responds to REGISTER requests and maintains a list of bindings that are accessible to proxy servers and redirect servers within its administrative domain. A registrar handles requests according to Section 8.2 and Section 17.2, but it accepts only REGISTER requests. A registrar MUST not generate 6xx responses.

A registrar MAY redirect REGISTER requests as appropriate. One common usage would be for a registrar listening on a multicast interface to redirect multicast REGISTER requests to its own unicast interface with a 302 (Moved Temporarily) response.

Registrars MUST ignore the Record-Route header field if it is included in a REGISTER request. Registrars MUST NOT include a Record-Route header field in any response to a REGISTER request.

A registrar might receive a request that traversed a proxy which treats REGISTER as an unknown request and which added a Record-Route header field value.

A registrar has to know (for example, through configuration) the set of domain(s) for which it maintains bindings. REGISTER requests MUST be processed by a registrar in the order that they are received. REGISTER requests MUST also be processed atomically, meaning that a particular REGISTER request is either processed completely or not at all. Each REGISTER message MUST be processed independently of any other registration or binding changes.

When receiving a REGISTER request, a registrar follows these steps:

1. The registrar inspects the Request-URI to determine whether it has access to bindings for the domain identified in the Request-URI. If not, and if the server also acts as a proxy server, the server SHOULD forward the request to the addressed domain, following the general behavior for proxying messages described in Section 16.

2. To guarantee that the registrar supports any necessary extensions, the registrar MUST process the Require header field values as described for UASs in Section 8.2.2.

3. A registrar SHOULD authenticate the UAC. Mechanisms for the authentication of SIP user agents are described in Section 22. Registration behavior in no way overrides the generic authentication framework for SIP. If no authentication mechanism is available, the registrar MAY take the From address as the asserted identity of the originator of the request.

4. The registrar SHOULD determine if the authenticated user is authorized to modify registrations for this address-of-record. For example, a registrar might consult an authorization database that maps user names to a list of addresses-of-record for which that user has authorization to modify bindings. If the authenticated user is not authorized to modify bindings, the registrar MUST return a 403 (Forbidden) and skip the remaining steps.

   In architectures that support third-party registration, one entity may be responsible for updating the registrations associated with multiple addresses-of-record.

5. The registrar extracts the address-of-record from the To header field of the request. If the address-of-record is not valid for the domain in the Request-URI, the registrar MUST send a 404 (Not Found) response and skip the remaining steps. The URI MUST then be converted to a canonical form. To do that, all URI parameters MUST be removed (including the user-param), and any escaped characters MUST be converted to their unescaped form. The result serves as an index into the list of bindings.

6. The registrar checks whether the request contains the Contact header field. If not, it skips to the last step. If the Contact header field is present, the registrar checks if there is one Contact field value that contains the special value "*" and an Expires field. If the request has additional Contact fields or an expiration time other than zero, the request is invalid, and the server MUST return a 400 (Invalid Request) and skip the remaining steps. If not, the registrar checks whether the Call-ID agrees with the value stored for each binding. If not, it MUST remove the binding. If it does agree, it MUST remove the binding only if the CSeq in the request is higher than the value stored for that binding. Otherwise, the update MUST be aborted and the request fails.

7. The registrar now processes each contact address in the Contact header field in turn. For each address, it determines the expiration interval as follows:

   - If the field value has an "expires" parameter, that value MUST be taken as the requested expiration.

   - If there is no such parameter, but the request has an Expires header field, that value MUST be taken as the requested expiration.

   - If there is neither, a locally-configured default value MUST be taken as the requested expiration.

   The registrar MAY choose an expiration less than the requested expiration interval. If and only if the requested expiration interval is greater than zero AND smaller than one hour AND less than a registrar-configured minimum, the registrar MAY reject the registration with a response of 423 (Interval Too Brief). This response MUST contain a Min-Expires header field that states the minimum expiration interval the registrar is willing to honor. It then skips the remaining steps.

   Allowing the registrar to set the registration interval protects it against excessively frequent registration refreshes while limiting the state that it needs to maintain and decreasing the likelihood of registrations going stale. The expiration interval of a registration is frequently used in the creation of services. An example is a follow-me service, where the user may only be available at a terminal for a brief period. Therefore, registrars should accept brief registrations; a request should only be rejected if the interval is so short that the refreshes would degrade registrar performance.

For each address, the registrar then searches the list of current bindings using the URI comparison rules. If the binding does not exist, it is tentatively added. If the binding does exist, the registrar checks the Call-ID value. If the Call-ID value in the existing binding differs from the Call-ID value in the request, the binding MUST be removed if the expiration time is zero and updated otherwise. If they are the same, the registrar compares the CSeq value. If the value is higher than that of the existing binding, it MUST update or remove the binding as above. If not, the update MUST be aborted and the request fails.

This algorithm ensures that out-of-order requests from the same UA are ignored.

Each binding record records the Call-ID and CSeq values from the request.

The binding updates MUST be committed (that is, made visible to the proxy or redirect server) if and only if all binding updates and additions succeed. If any one of them fails (for example, because the back-end database commit failed), the request MUST fail with a 500 (Server Error) response and all tentative binding updates MUST be removed.

8. The registrar returns a 200 (OK) response. The response MUST contain Contact header field values enumerating all current bindings. Each Contact value MUST feature an "expires" parameter indicating its expiration interval chosen by the registrar. The response SHOULD include a Date header field.

11 Querying for Capabilities

The SIP method OPTIONS allows a UA to query another UA or a proxy server as to its capabilities. This allows a client to discover information about the supported methods, content types, extensions, codecs, etc. without "ringing" the other party. For example, before a client inserts a Require header field into an INVITE listing an option that it is not certain the destination UAS supports, the client can query the destination UAS with an OPTIONS to see if this option is returned in a Supported header field. All UAs MUST support the OPTIONS method.

The target of the OPTIONS request is identified by the Request-URI, which could identify another UA or a SIP server. If the OPTIONS is addressed to a proxy server, the Request-URI is set without a user part, similar to the way a Request-URI is set for a REGISTER request.

Alternatively, a server receiving an OPTIONS request with a Max-Forwards header field value of 0 MAY respond to the request regardless of the Request-URI.

This behavior is common with HTTP/1.1. This behavior can be used as a "traceroute" functionality to check the capabilities of individual hop servers by sending a series of OPTIONS requests with incremented Max-Forwards values.

As is the case for general UA behavior, the transaction layer can return a timeout error if the OPTIONS yields no response. This may indicate that the target is unreachable and hence unavailable.

An OPTIONS request MAY be sent as part of an established dialog to query the peer on capabilities that may be utilized later in the dialog.

11.1 Construction of OPTIONS Request

An OPTIONS request is constructed using the standard rules for a SIP request as discussed in Section 8.1.1.

A Contact header field MAY be present in an OPTIONS.

An Accept header field SHOULD be included to indicate the type of message body the UAC wishes to receive in the response. Typically, this is set to a format that is used to describe the media capabilities of a UA, such as SDP (application/sdp).

The response to an OPTIONS request is assumed to be scoped to the Request-URI in the original request. However, only when an OPTIONS is sent as part of an established dialog is it guaranteed that future requests will be received by the server that generated the OPTIONS response.

Example OPTIONS request:

```
OPTIONS sip:carol@chicago.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
Max-Forwards: 70
To: <sip:carol@chicago.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:alice@pc33.atlanta.com>
Accept: application/sdp
Content-Length: 0
```

11.2 Processing of OPTIONS Request

The response to an OPTIONS is constructed using the standard rules for a SIP response as discussed in Section 8.2.6. The response code chosen MUST be the same that would have been chosen had the request been an INVITE. That is, a 200 (OK) would be returned if the UAS is ready to accept a call, a 486 (Busy Here) would be returned if the UAS is busy, etc. This allows an OPTIONS request to be used to determine the basic state of a UAS, which can be an indication of whether the UAS will accept an INVITE request.

An OPTIONS request received within a dialog generates a 200 (OK) response that is identical to one constructed outside a dialog and does not have any impact on the dialog.

This use of OPTIONS has limitations due to the differences in proxy handling of OPTIONS and INVITE requests. While a forked INVITE can result in multiple 200 (OK) responses being returned, a forked OPTIONS will only result in a single 200 (OK) response, since it is treated by proxies using the non-INVITE handling. See Section 16.7 for the normative details.

If the response to an OPTIONS is generated by a proxy server, the proxy returns a 200 (OK), listing the capabilities of the server. The response does not contain a message body.

Allow, Accept, Accept-Encoding, Accept-Language, and Supported header fields SHOULD be present in a 200 (OK) response to an OPTIONS request. If the response is generated by a proxy, the Allow header field SHOULD be omitted as it is ambiguous since a proxy is method agnostic. Contact header fields MAY be present in a 200 (OK) response and have the same semantics as in a 3xx response. That is, they may list a set of alternative names and methods of reaching the user. A Warning header field MAY be present.

A message body MAY be sent, the type of which is determined by the Accept header field in the OPTIONS request (application/sdp is the default if the Accept header field is not present). If the types include one that can describe media capabilities, the UAS SHOULD include a body in the response for that purpose. Details on the construction of such a body in the case of application/sdp are described in [13].

Example OPTIONS response generated by a UAS (corresponding to the request in Section 11.1):

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
 ;received=192.0.2.4
To: <sip:carol@chicago.com>;tag=93810874
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:carol@chicago.com>
Contact: <mailto:carol@chicago.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Content-Length: 274

(SDP not shown)
```

12 Dialogs

A key concept for a user agent is that of a dialog. A dialog represents a peer-to-peer SIP relationship between two user agents that persists for some time. The dialog facilitates sequencing of messages between the user agents and proper routing of requests between both of them. The dialog represents a context in which to interpret SIP messages. Section 8 discussed method independent UA processing for requests and responses outside of a dialog. This section discusses how those requests and responses are used to construct a dialog, and then how subsequent requests and responses are sent within a dialog.

A dialog is identified at each UA with a dialog ID, which consists of a Call-ID value, a local tag and a remote tag. The dialog ID at each UA involved in the dialog is not the same. Specifically, the local tag at one UA is identical to the remote tag at the peer UA. The tags are opaque tokens that facilitate the generation of unique dialog IDs.

A dialog ID is also associated with all responses and with any request that contains a tag in the To field. The rules for computing the dialog ID of a message depend on whether the SIP element is a UAC or UAS. For a UAC, the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote tag is set to the tag in the To field of the message, and the local tag is set to the tag in the From

field of the message (these rules apply to both requests and responses). As one would expect for a UAS, the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote tag is set to the tag in the From field of the message, and the local tag is set to the tag in the To field of the message.

A dialog contains certain pieces of state needed for further message transmissions within the dialog. This state consists of the dialog ID, a local sequence number (used to order requests from the UA to its peer), a remote sequence number (used to order requests from its peer to the UA), a local URI, a remote URI, remote target, a boolean flag called "secure", and a route set, which is an ordered list of URIs. The route set is the list of servers that need to be traversed to send a request to the peer. A dialog can also be in the "early" state, which occurs when it is created with a provisional response, and then transition to the "confirmed" state when a 2xx final response arrives. For other responses, or if no response arrives at all on that dialog, the early dialog terminates.

12.1 Creation of a Dialog

Dialogs are created through the generation of non-failure responses to requests with specific methods. Within this specification, only 2xx and 101-199 responses with a To tag, where the request was INVITE, will establish a dialog. A dialog established by a non-final response to a request is in the "early" state and it is called an early dialog. Extensions MAY define other means for creating dialogs. Section 13 gives more details that are specific to the INVITE method. Here, we describe the process for creation of dialog state that is not dependent on the method.

UAs MUST assign values to the dialog ID components as described below.

12.1.1 UAS behavior

When a UAS responds to a request with a response that establishes a dialog (such as a 2xx to INVITE), the UAS MUST copy all Record-Route header field values from the request into the response (including the URIs, URI parameters, and any Record-Route header field parameters, whether they are known or unknown to the UAS) and MUST maintain the order of those values. The UAS MUST add a Contact header field to the response. The Contact header field contains an address where the UAS would like to be contacted for subsequent requests in the dialog (which includes the ACK for a 2xx response in the case of an INVITE). Generally, the host portion of this URI is the IP address or FQDN of the host. The URI provided in the Contact header field MUST be a SIP or SIPS URI. If the request that initiated the dialog contained a

SIPS URI in the Request-URI or in the top Record-Route header field value, if there was any, or the Contact header field if there was no Record-Route header field, the Contact header field in the response MUST be a SIPS URI. The URI SHOULD have global scope (that is, the same URI can be used in messages outside this dialog). The same way, the scope of the URI in the Contact header field of the INVITE is not limited to this dialog either. It can therefore be used in messages to the UAC even outside this dialog.

The UAS then constructs the state of the dialog. This state MUST be maintained for the duration of the dialog.

If the request arrived over TLS, and the Request-URI contained a SIPS URI, the "secure" flag is set to TRUE.

The route set MUST be set to the list of URIs in the Record-Route header field from the request, taken in order and preserving all URI parameters. If no Record-Route header field is present in the request, the route set MUST be set to the empty set. This route set, even if empty, overrides any pre-existing route set for future requests in this dialog. The remote target MUST be set to the URI from the Contact header field of the request.

The remote sequence number MUST be set to the value of the sequence number in the CSeq header field of the request. The local sequence number MUST be empty. The call identifier component of the dialog ID MUST be set to the value of the Call-ID in the request. The local tag component of the dialog ID MUST be set to the tag in the To field in the response to the request (which always includes a tag), and the remote tag component of the dialog ID MUST be set to the tag from the From field in the request. A UAS MUST be prepared to receive a request without a tag in the From field, in which case the tag is considered to have a value of null.

> This is to maintain backwards compatibility with RFC 2543, which did not mandate From tags.

The remote URI MUST be set to the URI in the From field, and the local URI MUST be set to the URI in the To field.

12.1.2 UAC Behavior

When a UAC sends a request that can establish a dialog (such as an INVITE) it MUST provide a SIP or SIPS URI with global scope (i.e., the same SIP URI can be used in messages outside this dialog) in the Contact header field of the request. If the request has a Request-URI or a topmost Route header field value with a SIPS URI, the Contact header field MUST contain a SIPS URI.

When a UAC receives a response that establishes a dialog, it constructs the state of the dialog. This state MUST be maintained for the duration of the dialog.

If the request was sent over TLS, and the Request-URI contained a SIPS URI, the "secure" flag is set to TRUE.

The route set MUST be set to the list of URIs in the Record-Route header field from the response, taken in reverse order and preserving all URI parameters. If no Record-Route header field is present in the response, the route set MUST be set to the empty set. This route set, even if empty, overrides any pre-existing route set for future requests in this dialog. The remote target MUST be set to the URI from the Contact header field of the response.

The local sequence number MUST be set to the value of the sequence number in the CSeq header field of the request. The remote sequence number MUST be empty (it is established when the remote UA sends a request within the dialog). The call identifier component of the dialog ID MUST be set to the value of the Call-ID in the request. The local tag component of the dialog ID MUST be set to the tag in the From field in the request, and the remote tag component of the dialog ID MUST be set to the tag in the To field of the response. A UAC MUST be prepared to receive a response without a tag in the To field, in which case the tag is considered to have a value of null.

> This is to maintain backwards compatibility with RFC 2543, which did not mandate To tags.

The remote URI MUST be set to the URI in the To field, and the local URI MUST be set to the URI in the From field.

12.2 Requests within a Dialog

Once a dialog has been established between two UAs, either of them MAY initiate new transactions as needed within the dialog. The UA sending the request will take the UAC role for the transaction. The UA receiving the request will take the UAS role. Note that these may be different roles than the UAs held during the transaction that established the dialog.

Requests within a dialog MAY contain Record-Route and Contact header fields. However, these requests do not cause the dialog's route set to be modified, although they may modify the remote target URI. Specifically, requests that are not target refresh requests do not modify the dialog's remote target URI, and requests that are target refresh requests do. For dialogs that have been established with an

INVITE, the only target refresh request defined is re-INVITE (see Section 14). Other extensions may define different target refresh requests for dialogs established in other ways.

Note that an ACK is NOT a target refresh request.

Target refresh requests only update the dialog's remote target URI, and not the route set formed from the Record-Route. Updating the latter would introduce severe backwards compatibility problems with RFC 2543-compliant systems.

12.2.1 UAC Behavior

12.2.1.1 Generating the Request

A request within a dialog is constructed by using many of the components of the state stored as part of the dialog.

The URI in the To field of the request MUST be set to the remote URI from the dialog state. The tag in the To header field of the request MUST be set to the remote tag of the dialog ID. The From URI of the request MUST be set to the local URI from the dialog state. The tag in the From header field of the request MUST be set to the local tag of the dialog ID. If the value of the remote or local tags is null, the tag parameter MUST be omitted from the To or From header fields, respectively.

Usage of the URI from the To and From fields in the original request within subsequent requests is done for backwards compatibility with RFC 2543, which used the URI for dialog identification. In this specification, only the tags are used for dialog identification. It is expected that mandatory reflection of the original To and From URI in mid-dialog requests will be deprecated in a subsequent revision of this specification.

The Call-ID of the request MUST be set to the Call-ID of the dialog. Requests within a dialog MUST contain strictly monotonically increasing and contiguous CSeq sequence numbers (increasing-by-one) in each direction (excepting ACK and CANCEL of course, whose numbers equal the requests being acknowledged or cancelled). Therefore, if the local sequence number is not empty, the value of the local sequence number MUST be incremented by one, and this value MUST be placed into the CSeq header field. If the local sequence number is empty, an initial value MUST be chosen using the guidelines of Section 8.1.1.5. The method field in the CSeq header field value MUST match the method of the request.

With a length of 32 bits, a client could generate, within a single call, one request a second for about 136 years before needing to wrap around. The initial value of the sequence number is chosen so that subsequent requests within the same call will not wrap around. A non-zero initial value allows clients to use a time-based initial sequence number. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial sequence number.

The UAC uses the remote target and route set to build the Request-URI and Route header field of the request.

If the route set is empty, the UAC MUST place the remote target URI into the Request-URI. The UAC MUST NOT add a Route header field to the request.

If the route set is not empty, and the first URI in the route set contains the lr parameter (see Section 19.1.1), the UAC MUST place the remote target URI into the Request-URI and MUST include a Route header field containing the route set values in order, including all parameters.

If the route set is not empty, and its first URI does not contain the lr parameter, the UAC MUST place the first URI from the route set into the Request-URI, stripping any parameters that are not allowed in a Request-URI. The UAC MUST add a Route header field containing the remainder of the route set values in order, including all parameters. The UAC MUST then place the remote target URI into the Route header field as the last value.

For example, if the remote target is sip:user@remoteua and the route set contains:

```
<sip:proxy1>,<sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>
```

The request will be formed with the following Request-URI and Route header field:

```
METHOD sip:proxy1
Route: <sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>,<sip:user@remoteua>
```

If the first URI of the route set does not contain the lr parameter, the proxy indicated does not understand the routing mechanisms described in this document and will act as specified in RFC 2543, replacing the Request-URI with the first Route header field value it receives while forwarding the message. Placing the Request-URI at the end of the Route header field preserves the

information in that Request-URI across the strict router (it will be returned to the Request-URI when the request reaches a loose-router).

A UAC SHOULD include a Contact header field in any target refresh requests within a dialog, and unless there is a need to change it, the URI SHOULD be the same as used in previous requests within the dialog. If the "secure" flag is true, that URI MUST be a SIPS URI. As discussed in Section 12.2.2, a Contact header field in a target refresh request updates the remote target URI. This allows a UA to provide a new contact address, should its address change during the duration of the dialog.

However, requests that are not target refresh requests do not affect the remote target URI for the dialog.

The rest of the request is formed as described in Section 8.1.1.

Once the request has been constructed, the address of the server is computed and the request is sent, using the same procedures for requests outside of a dialog (Section 8.1.2).

The procedures in Section 8.1.2 will normally result in the request being sent to the address indicated by the topmost Route header field value or the Request-URI if no Route header field is present. Subject to certain restrictions, they allow the request to be sent to an alternate address (such as a default outbound proxy not represented in the route set).

12.2.1.2 Processing the Responses

The UAC will receive responses to the request from the transaction layer. If the client transaction returns a timeout, this is treated as a 408 (Request Timeout) response.

The behavior of a UAC that receives a 3xx response for a request sent within a dialog is the same as if the request had been sent outside a dialog. This behavior is described in Section 8.1.3.4.

Note, however, that when the UAC tries alternative locations, it still uses the route set for the dialog to build the Route header of the request.

When a UAC receives a 2xx response to a target refresh request, it MUST replace the dialog's remote target URI with the URI from the Contact header field in that response, if present.

If the response for a request within a dialog is a 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout), the UAC SHOULD terminate the dialog. A UAC SHOULD also terminate a dialog if no response at all is received for the request (the client transaction would inform the TU about the timeout.)

For INVITE initiated dialogs, terminating the dialog consists of sending a BYE.

12.2.2 UAS Behavior

Requests sent within a dialog, as any other requests, are atomic. If a particular request is accepted by the UAS, all the state changes associated with it are performed. If the request is rejected, none of the state changes are performed.

Note that some requests, such as INVITEs, affect several pieces of state.

The UAS will receive the request from the transaction layer. If the request has a tag in the To header field, the UAS core computes the dialog identifier corresponding to the request and compares it with existing dialogs. If there is a match, this is a mid-dialog request. In that case, the UAS first applies the same processing rules for requests outside of a dialog, discussed in Section 8.2.

If the request has a tag in the To header field, but the dialog identifier does not match any existing dialogs, the UAS may have crashed and restarted, or it may have received a request for a different (possibly failed) UAS (the UASs can construct the To tags so that a UAS can identify that the tag was for a UAS for which it is providing recovery). Another possibility is that the incoming request has been simply misrouted. Based on the To tag, the UAS MAY either accept or reject the request. Accepting the request for acceptable To tags provides robustness, so that dialogs can persist even through crashes. UAs wishing to support this capability must take into consideration some issues such as choosing monotonically increasing CSeq sequence numbers even across reboots, reconstructing the route set, and accepting out-of-range RTP timestamps and sequence numbers.

If the UAS wishes to reject the request because it does not wish to recreate the dialog, it MUST respond to the request with a 481 (Call/Transaction Does Not Exist) status code and pass that to the server transaction.

Requests that do not change in any way the state of a dialog may be received within a dialog (for example, an OPTIONS request). They are processed as if they had been received outside the dialog.

If the remote sequence number is empty, it MUST be set to the value of the sequence number in the CSeq header field value in the request. If the remote sequence number was not empty, but the sequence number of the request is lower than the remote sequence number, the request is out of order and MUST be rejected with a 500 (Server Internal Error) response. If the remote sequence number was not empty, and the sequence number of the request is greater than the remote sequence number, the request is in order. It is possible for the CSeq sequence number to be higher than the remote sequence number by more than one. This is not an error condition, and a UAS SHOULD be prepared to receive and process requests with CSeq values more than one higher than the previous received request. The UAS MUST then set the remote sequence number to the value of the sequence number in the CSeq header field value in the request.

> If a proxy challenges a request generated by the UAC, the UAC has to resubmit the request with credentials. The resubmitted request will have a new CSeq number. The UAS will never see the first request, and thus, it will notice a gap in the CSeq number space. Such a gap does not represent any error condition.

When a UAS receives a target refresh request, it MUST replace the dialog's remote target URI with the URI from the Contact header field in that request, if present.

12.3 Termination of a Dialog

Independent of the method, if a request outside of a dialog generates a non-2xx final response, any early dialogs created through provisional responses to that request are terminated. The mechanism for terminating confirmed dialogs is method specific. In this specification, the BYE method terminates a session and the dialog associated with it. See Section 15 for details.

13 Initiating a Session

13.1 Overview

When a user agent client desires to initiate a session (for example, audio, video, or a game), it formulates an INVITE request. The INVITE request asks a server to establish a session. This request may be forwarded by proxies, eventually arriving at one or more UAS that can potentially accept the invitation. These UASs will frequently need to query the user about whether to accept the

invitation. After some time, those UASs can accept the invitation (meaning the session is to be established) by sending a 2xx response. If the invitation is not accepted, a 3xx, 4xx, 5xx or 6xx response is sent, depending on the reason for the rejection. Before sending a final response, the UAS can also send provisional responses (1xx) to advise the UAC of progress in contacting the called user.

After possibly receiving one or more provisional responses, the UAC will get one or more 2xx responses or one non-2xx final response. Because of the protracted amount of time it can take to receive final responses to INVITE, the reliability mechanisms for INVITE transactions differ from those of other requests (like OPTIONS). Once it receives a final response, the UAC needs to send an ACK for every final response it receives. The procedure for sending this ACK depends on the type of response. For final responses between 300 and 699, the ACK processing is done in the transaction layer and follows one set of rules (See Section 17). For 2xx responses, the ACK is generated by the UAC core.

A 2xx response to an INVITE establishes a session, and it also creates a dialog between the UA that issued the INVITE and the UA that generated the 2xx response. Therefore, when multiple 2xx responses are received from different remote UAs (because the INVITE forked), each 2xx establishes a different dialog. All these dialogs are part of the same call.

This section provides details on the establishment of a session using INVITE. A UA that supports INVITE MUST also support ACK, CANCEL and BYE.

13.2 UAC Processing

13.2.1 Creating the Initial INVITE

Since the initial INVITE represents a request outside of a dialog, its construction follows the procedures of Section 8.1.1. Additional processing is required for the specific case of INVITE.

An Allow header field (Section 20.5) SHOULD be present in the INVITE. It indicates what methods can be invoked within a dialog, on the UA sending the INVITE, for the duration of the dialog. For example, a UA capable of receiving INFO requests within a dialog [34] SHOULD include an Allow header field listing the INFO method.

A Supported header field (Section 20.37) SHOULD be present in the INVITE. It enumerates all the extensions understood by the UAC.

An Accept (Section 20.1) header field MAY be present in the INVITE. It indicates which Content-Types are acceptable to the UA, in both the response received by it, and in any subsequent requests sent to it within dialogs established by the INVITE. The Accept header field is especially useful for indicating support of various session description formats.

The UAC MAY add an Expires header field (Section 20.19) to limit the validity of the invitation. If the time indicated in the Expires header field is reached and no final answer for the INVITE has been received, the UAC core SHOULD generate a CANCEL request for the INVITE, as per Section 9.

A UAC MAY also find it useful to add, among others, Subject (Section 20.36), Organization (Section 20.25) and User-Agent (Section 20.41) header fields. They all contain information related to the INVITE.

The UAC MAY choose to add a message body to the INVITE. Section 8.1.1.10 deals with how to construct the header fields -- Content-Type among others -- needed to describe the message body.

There are special rules for message bodies that contain a session description - their corresponding Content-Disposition is "session". SIP uses an offer/answer model where one UA sends a session description, called the offer, which contains a proposed description of the session. The offer indicates the desired communications means (audio, video, games), parameters of those means (such as codec types) and addresses for receiving media from the answerer. The other UA responds with another session description, called the answer, which indicates which communications means are accepted, the parameters that apply to those means, and addresses for receiving media from the offerer. An offer/answer exchange is within the context of a dialog, so that if a SIP INVITE results in multiple dialogs, each is a separate offer/answer exchange. The offer/answer model defines restrictions on when offers and answers can be made (for example, you cannot make a new offer while one is in progress). This results in restrictions on where the offers and answers can appear in SIP messages. In this specification, offers and answers can only appear in INVITE requests and responses, and ACK. The usage of offers and answers is further restricted. For the initial INVITE transaction, the rules are:

- The initial offer MUST be in either an INVITE or, if not there, in the first reliable non-failure message from the UAS back to the UAC. In this specification, that is the final 2xx response.

- o If the initial offer is in an INVITE, the answer MUST be in a reliable non-failure message from UAS back to UAC which is correlated to that INVITE. For this specification, that is only the final 2xx response to that INVITE. That same exact answer MAY also be placed in any provisional responses sent prior to the answer. The UAC MUST treat the first session description it receives as the answer, and MUST ignore any session descriptions in subsequent responses to the initial INVITE.

- o If the initial offer is in the first reliable non-failure message from the UAS back to UAC, the answer MUST be in the acknowledgement for that message (in this specification, ACK for a 2xx response).

- o After having sent or received an answer to the first offer, the UAC MAY generate subsequent offers in requests based on rules specified for that method, but only if it has received answers to any previous offers, and has not sent any offers to which it hasn't gotten an answer.

- o Once the UAS has sent or received an answer to the initial offer, it MUST NOT generate subsequent offers in any responses to the initial INVITE. This means that a UAS based on this specification alone can never generate subsequent offers until completion of the initial transaction.

Concretely, the above rules specify two exchanges for UAs compliant to this specification alone - the offer is in the INVITE, and the answer in the 2xx (and possibly in a 1xx as well, with the same value), or the offer is in the 2xx, and the answer is in the ACK. All user agents that support INVITE MUST support these two exchanges.

The Session Description Protocol (SDP) (RFC 2327 [1]) MUST be supported by all user agents as a means to describe sessions, and its usage for constructing offers and answers MUST follow the procedures defined in [13].

The restrictions of the offer-answer model just described only apply to bodies whose Content-Disposition header field value is "session". Therefore, it is possible that both the INVITE and the ACK contain a body message (for example, the INVITE carries a photo (Content-Disposition: render) and the ACK a session description (Content-Disposition: session)).

If the Content-Disposition header field is missing, bodies of Content-Type application/sdp imply the disposition "session", while other content types imply "render".

Once the INVITE has been created, the UAC follows the procedures defined for sending requests outside of a dialog (Section 8). This results in the construction of a client transaction that will ultimately send the request and deliver responses to the UAC.

13.2.2 Processing INVITE Responses

Once the INVITE has been passed to the INVITE client transaction, the UAC waits for responses for the INVITE. If the INVITE client transaction returns a timeout rather than a response the TU acts as if a 408 (Request Timeout) response had been received, as described in Section 8.1.3.

13.2.2.1 1xx Responses

Zero, one or multiple provisional responses may arrive before one or more final responses are received. Provisional responses for an INVITE request can create "early dialogs". If a provisional response has a tag in the To field, and if the dialog ID of the response does not match an existing dialog, one is constructed using the procedures defined in Section 12.1.2.

The early dialog will only be needed if the UAC needs to send a request to its peer within the dialog before the initial INVITE transaction completes. Header fields present in a provisional response are applicable as long as the dialog is in the early state (for example, an Allow header field in a provisional response contains the methods that can be used in the dialog while this is in the early state).

13.2.2.2 3xx Responses

A 3xx response may contain one or more Contact header field values providing new addresses where the callee might be reachable. Depending on the status code of the 3xx response (see Section 21.3), the UAC MAY choose to try those new addresses.

13.2.2.3 4xx, 5xx and 6xx Responses

A single non-2xx final response may be received for the INVITE. 4xx, 5xx and 6xx responses may contain a Contact header field value indicating the location where additional information about the error can be found. Subsequent final responses (which would only arrive under error conditions) MUST be ignored.

All early dialogs are considered terminated upon reception of the non-2xx final response.

After having received the non-2xx final response the UAC core considers the INVITE transaction completed. The INVITE client transaction handles the generation of ACKs for the response (see Section 17).

13.2.2.4 2xx Responses

Multiple 2xx responses may arrive at the UAC for a single INVITE request due to a forking proxy. Each response is distinguished by the tag parameter in the To header field, and each represents a distinct dialog, with a distinct dialog identifier.

If the dialog identifier in the 2xx response matches the dialog identifier of an existing dialog, the dialog MUST be transitioned to the "confirmed" state, and the route set for the dialog MUST be recomputed based on the 2xx response using the procedures of Section 12.2.1.2. Otherwise, a new dialog in the "confirmed" state MUST be constructed using the procedures of Section 12.1.2.

> Note that the only piece of state that is recomputed is the route set. Other pieces of state such as the highest sequence numbers (remote and local) sent within the dialog are not recomputed. The route set only is recomputed for backwards compatibility. RFC 2543 did not mandate mirroring of the Record-Route header field in a 1xx, only 2xx. However, we cannot update the entire state of the dialog, since mid-dialog requests may have been sent within the early dialog, modifying the sequence numbers, for example.

The UAC core MUST generate an ACK request for each 2xx received from the transaction layer. The header fields of the ACK are constructed in the same way as for any request sent within a dialog (see Section 12) with the exception of the CSeq and the header fields related to authentication. The sequence number of the CSeq header field MUST be the same as the INVITE being acknowledged, but the CSeq method MUST be ACK. The ACK MUST contain the same credentials as the INVITE. If the 2xx contains an offer (based on the rules above), the ACK MUST carry an answer in its body. If the offer in the 2xx response is not acceptable, the UAC core MUST generate a valid answer in the ACK and then send a BYE immediately.

Once the ACK has been constructed, the procedures of [4] are used to determine the destination address, port and transport. However, the request is passed to the transport layer directly for transmission, rather than a client transaction. This is because the UAC core handles retransmissions of the ACK, not the transaction layer. The ACK MUST be passed to the client transport every time a retransmission of the 2xx final response that triggered the ACK arrives.

The UAC core considers the INVITE transaction completed 64*T1 seconds after the reception of the first 2xx response. At this point all the early dialogs that have not transitioned to established dialogs are terminated. Once the INVITE transaction is considered completed by the UAC core, no more new 2xx responses are expected to arrive.

If, after acknowledging any 2xx response to an INVITE, the UAC does not want to continue with that dialog, then the UAC MUST terminate the dialog by sending a BYE request as described in Section 15.

13.3 UAS Processing

13.3.1 Processing of the INVITE

The UAS core will receive INVITE requests from the transaction layer. It first performs the request processing procedures of Section 8.2, which are applied for both requests inside and outside of a dialog.

Assuming these processing states are completed without generating a response, the UAS core performs the additional processing steps:

1. If the request is an INVITE that contains an Expires header field, the UAS core sets a timer for the number of seconds indicated in the header field value. When the timer fires, the invitation is considered to be expired. If the invitation expires before the UAS has generated a final response, a 487 (Request Terminated) response SHOULD be generated.

2. If the request is a mid-dialog request, the method-independent processing described in Section 12.2.2 is first applied. It might also modify the session; Section 14 provides details.

3. If the request has a tag in the To header field but the dialog identifier does not match any of the existing dialogs, the UAS may have crashed and restarted, or may have received a request for a different (possibly failed) UAS. Section 12.2.2 provides guidelines to achieve a robust behavior under such a situation.

Processing from here forward assumes that the INVITE is outside of a dialog, and is thus for the purposes of establishing a new session.

The INVITE may contain a session description, in which case the UAS is being presented with an offer for that session. It is possible that the user is already a participant in that session, even though the INVITE is outside of a dialog. This can happen when a user is invited to the same multicast conference by multiple other participants. If desired, the UAS MAY use identifiers within the session description to detect this duplication. For example, SDP

contains a session id and version number in the origin (o) field. If the user is already a member of the session, and the session parameters contained in the session description have not changed, the UAS MAY silently accept the INVITE (that is, send a 2xx response without prompting the user).

If the INVITE does not contain a session description, the UAS is being asked to participate in a session, and the UAC has asked that the UAS provide the offer of the session. It MUST provide the offer in its first non-failure reliable message back to the UAC. In this specification, that is a 2xx response to the INVITE.

The UAS can indicate progress, accept, redirect, or reject the invitation. In all of these cases, it formulates a response using the procedures described in Section 8.2.6.

13.3.1.1 Progress

If the UAS is not able to answer the invitation immediately, it can choose to indicate some kind of progress to the UAC (for example, an indication that a phone is ringing). This is accomplished with a provisional response between 101 and 199. These provisional responses establish early dialogs and therefore follow the procedures of Section 12.1.1 in addition to those of Section 8.2.6. A UAS MAY send as many provisional responses as it likes. Each of these MUST indicate the same dialog ID. However, these will not be delivered reliably.

If the UAS desires an extended period of time to answer the INVITE, it will need to ask for an "extension" in order to prevent proxies from canceling the transaction. A proxy has the option of canceling a transaction when there is a gap of 3 minutes between responses in a transaction. To prevent cancellation, the UAS MUST send a non-100 provisional response at every minute, to handle the possibility of lost provisional responses.

> An INVITE transaction can go on for extended durations when the user is placed on hold, or when interworking with PSTN systems which allow communications to take place without answering the call. The latter is common in Interactive Voice Response (IVR) systems.

13.3.1.2 The INVITE is Redirected

If the UAS decides to redirect the call, a 3xx response is sent. A 300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain a Contact header field

containing one or more URIs of new addresses to be tried. The response is passed to the INVITE server transaction, which will deal with its retransmissions.

13.3.1.3 The INVITE is Rejected

A common scenario occurs when the callee is currently not willing or able to take additional calls at this end system. A 486 (Busy Here) SHOULD be returned in such a scenario. If the UAS knows that no other end system will be able to accept this call, a 600 (Busy Everywhere) response SHOULD be sent instead. However, it is unlikely that a UAS will be able to know this in general, and thus this response will not usually be used. The response is passed to the INVITE server transaction, which will deal with its retransmissions.

A UAS rejecting an offer contained in an INVITE SHOULD return a 488 (Not Acceptable Here) response. Such a response SHOULD include a Warning header field value explaining why the offer was rejected.

13.3.1.4 The INVITE is Accepted

The UAS core generates a 2xx response. This response establishes a dialog, and therefore follows the procedures of Section 12.1.1 in addition to those of Section 8.2.6.

A 2xx response to an INVITE SHOULD contain the Allow header field and the Supported header field, and MAY contain the Accept header field. Including these header fields allows the UAC to determine the features and extensions supported by the UAS for the duration of the call, without probing.

If the INVITE request contained an offer, and the UAS had not yet sent an answer, the 2xx MUST contain an answer. If the INVITE did not contain an offer, the 2xx MUST contain an offer if the UAS had not yet sent an offer.

Once the response has been constructed, it is passed to the INVITE server transaction. Note, however, that the INVITE server transaction will be destroyed as soon as it receives this final response and passes it to the transport. Therefore, it is necessary to periodically pass the response directly to the transport until the ACK arrives. The 2xx response is passed to the transport with an interval that starts at T1 seconds and doubles for each retransmission until it reaches T2 seconds (T1 and T2 are defined in Section 17). Response retransmissions cease when an ACK request for the response is received. This is independent of whatever transport protocols are used to send the response.

Since 2xx is retransmitted end-to-end, there may be hops between UAS and UAC that are UDP. To ensure reliable delivery across these hops, the response is retransmitted periodically even if the transport at the UAS is reliable.

If the server retransmits the 2xx response for 64*T1 seconds without receiving an ACK, the dialog is confirmed, but the session SHOULD be terminated. This is accomplished with a BYE, as described in Section 15.

14 Modifying an Existing Session

A successful INVITE request (see Section 13) establishes both a dialog between two user agents and a session using the offer-answer model. Section 12 explains how to modify an existing dialog using a target refresh request (for example, changing the remote target URI of the dialog). This section describes how to modify the actual session. This modification can involve changing addresses or ports, adding a media stream, deleting a media stream, and so on. This is accomplished by sending a new INVITE request within the same dialog that established the session. An INVITE request sent within an existing dialog is known as a re-INVITE.

Note that a single re-INVITE can modify the dialog and the parameters of the session at the same time.

Either the caller or callee can modify an existing session.

The behavior of a UA on detection of media failure is a matter of local policy. However, automated generation of re-INVITE or BYE is NOT RECOMMENDED to avoid flooding the network with traffic when there is congestion. In any case, if these messages are sent automatically, they SHOULD be sent after some randomized interval.

Note that the paragraph above refers to automatically generated BYEs and re-INVITEs. If the user hangs up upon media failure, the UA would send a BYE request as usual.

14.1 UAC Behavior

The same offer-answer model that applies to session descriptions in INVITEs (Section 13.2.1) applies to re-INVITEs. As a result, a UAC that wants to add a media stream, for example, will create a new offer that contains this media stream, and send that in an INVITE request to its peer. It is important to note that the full description of the session, not just the change, is sent. This supports stateless session processing in various elements, and supports failover and recovery capabilities. Of course, a UAC MAY

send a re-INVITE with no session description, in which case the first reliable non-failure response to the re-INVITE will contain the offer (in this specification, that is a 2xx response).

If the session description format has the capability for version numbers, the offerer SHOULD indicate that the version of the session description has changed.

The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set following the same rules as for regular requests within an existing dialog, described in Section 12.

A UAC MAY choose not to add an Alert-Info header field or a body with Content-Disposition "alert" to re-INVITEs because UASs do not typically alert the user upon reception of a re-INVITE.

Unlike an INVITE, which can fork, a re-INVITE will never fork, and therefore, only ever generate a single final response. The reason a re-INVITE will never fork is that the Request-URI identifies the target as the UA instance it established the dialog with, rather than identifying an address-of-record for the user.

Note that a UAC MUST NOT initiate a new INVITE transaction within a dialog while another INVITE transaction is in progress in either direction.

1. If there is an ongoing INVITE client transaction, the TU MUST wait until the transaction reaches the completed or terminated state before initiating the new INVITE.

2. If there is an ongoing INVITE server transaction, the TU MUST wait until the transaction reaches the confirmed or terminated state before initiating the new INVITE.

However, a UA MAY initiate a regular transaction while an INVITE transaction is in progress. A UA MAY also initiate an INVITE transaction while a regular transaction is in progress.

If a UA receives a non-2xx final response to a re-INVITE, the session parameters MUST remain unchanged, as if no re-INVITE had been issued. Note that, as stated in Section 12.2.1.2, if the non-2xx final response is a 481 (Call/Transaction Does Not Exist), or a 408 (Request Timeout), or no response at all is received for the re-INVITE (that is, a timeout is returned by the INVITE client transaction), the UAC will terminate the dialog.

If a UAC receives a 491 response to a re-INVITE, it SHOULD start a timer with a value T chosen as follows:

1. If the UAC is the owner of the Call-ID of the dialog ID (meaning it generated the value), T has a randomly chosen value between 2.1 and 4 seconds in units of 10 ms.

2. If the UAC is not the owner of the Call-ID of the dialog ID, T has a randomly chosen value of between 0 and 2 seconds in units of 10 ms.

When the timer fires, the UAC SHOULD attempt the re-INVITE once more, if it still desires for that session modification to take place. For example, if the call was already hung up with a BYE, the re-INVITE would not take place.

The rules for transmitting a re-INVITE and for generating an ACK for a 2xx response to re-INVITE are the same as for the initial INVITE (Section 13.2.1).

14.2 UAS Behavior

Section 13.3.1 describes the procedure for distinguishing incoming re-INVITEs from incoming initial INVITEs and handling a re-INVITE for an existing dialog.

A UAS that receives a second INVITE before it sends the final response to a first INVITE with a lower CSeq sequence number on the same dialog MUST return a 500 (Server Internal Error) response to the second INVITE and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds.

A UAS that receives an INVITE on a dialog while an INVITE it had sent on that dialog is in progress MUST return a 491 (Request Pending) response to the received INVITE.

If a UA receives a re-INVITE for an existing dialog, it MUST check any version identifiers in the session description or, if there are no version identifiers, the content of the session description to see if it has changed. If the session description has changed, the UAS MUST adjust the session parameters accordingly, possibly after asking the user for confirmation.

Versioning of the session description can be used to accommodate the capabilities of new arrivals to a conference, add or delete media, or change from a unicast to a multicast conference.

If the new session description is not acceptable, the UAS can reject it by returning a 488 (Not Acceptable Here) response for the re-INVITE. This response SHOULD include a Warning header field.

If a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a BYE to terminate the dialog.

A UAS MAY choose not to generate 180 (Ringing) responses for a re-INVITE because UACs do not typically render this information to the user. For the same reason, UASs MAY choose not to use an Alert-Info header field or a body with Content-Disposition "alert" in responses to a re-INVITE.

A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) SHOULD construct the offer as if the UAS were making a brand new call, subject to the constraints of sending an offer that updates an existing session, as described in [13] in the case of SDP. Specifically, this means that it SHOULD include as many media formats and media types that the UA is willing to support. The UAS MUST ensure that the session description overlaps with its previous session description in media formats, transports, or other parameters that require support from the peer. This is to avoid the need for the peer to reject the session description. If, however, it is unacceptable to the UAC, the UAC SHOULD generate an answer with a valid session description, and then send a BYE to terminate the session.

15 Terminating a Session

This section describes the procedures for terminating a session established by SIP. The state of the session and the state of the dialog are very closely related. When a session is initiated with an INVITE, each 1xx or 2xx response from a distinct UAS creates a dialog, and if that response completes the offer/answer exchange, it also creates a session. As a result, each session is "associated" with a single dialog - the one which resulted in its creation. If an initial INVITE generates a non-2xx final response, that terminates all sessions (if any) and all dialogs (if any) that were created through responses to the request. By virtue of completing the transaction, a non-2xx final response also prevents further sessions from being created as a result of the INVITE. The BYE request is used to terminate a specific session or attempted session. In this case, the specific session is the one with the peer UA on the other side of the dialog. When a BYE is received on a dialog, any session associated with that dialog SHOULD terminate. A UA MUST NOT send a BYE outside of a dialog. The caller's UA MAY send a BYE for either confirmed or early dialogs, and the callee's UA MAY send a BYE on confirmed dialogs, but MUST NOT send a BYE on early dialogs.

However, the callee's UA MUST NOT send a BYE on a confirmed dialog until it has received an ACK for its 2xx response or until the server transaction times out. If no SIP extensions have defined other application layer states associated with the dialog, the BYE also terminates the dialog.

The impact of a non-2xx final response to INVITE on dialogs and sessions makes the use of CANCEL attractive. The CANCEL attempts to force a non-2xx response to the INVITE (in particular, a 487). Therefore, if a UAC wishes to give up on its call attempt entirely, it can send a CANCEL. If the INVITE results in 2xx final response(s) to the INVITE, this means that a UAS accepted the invitation while the CANCEL was in progress. The UAC MAY continue with the sessions established by any 2xx responses, or MAY terminate them with BYE.

> The notion of "hanging up" is not well defined within SIP. It is specific to a particular, albeit common, user interface. Typically, when the user hangs up, it indicates a desire to terminate the attempt to establish a session, and to terminate any sessions already created. For the caller's UA, this would imply a CANCEL request if the initial INVITE has not generated a final response, and a BYE to all confirmed dialogs after a final response. For the callee's UA, it would typically imply a BYE; presumably, when the user picked up the phone, a 2xx was generated, and so hanging up would result in a BYE after the ACK is received. This does not mean a user cannot hang up before receipt of the ACK, it just means that the software in his phone needs to maintain state for a short while in order to clean up properly. If the particular UI allows for the user to reject a call before its answered, a 403 (Forbidden) is a good way to express that. As per the rules above, a BYE can't be sent.

15.1 Terminating a Session with a BYE Request

15.1.1 UAC Behavior

A BYE request is constructed as would any other request within a dialog, as described in Section 12.

Once the BYE is constructed, the UAC core creates a new non-INVITE client transaction, and passes it the BYE request. The UAC MUST consider the session terminated (and therefore stop sending or listening for media) as soon as the BYE request is passed to the client transaction. If the response for the BYE is a 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout) or no

response at all is received for the BYE (that is, a timeout is returned by the client transaction), the UAC MUST consider the session and the dialog terminated.

15.1.2 UAS Behavior

A UAS first processes the BYE request according to the general UAS processing described in Section 8.2. A UAS core receiving a BYE request checks if it matches an existing dialog. If the BYE does not match an existing dialog, the UAS core SHOULD generate a 481 (Call/Transaction Does Not Exist) response and pass that to the server transaction.

This rule means that a BYE sent without tags by a UAC will be rejected. This is a change from RFC 2543, which allowed BYE without tags.

A UAS core receiving a BYE request for an existing dialog MUST follow the procedures of Section 12.2.2 to process the request. Once done, the UAS SHOULD terminate the session (and therefore stop sending and listening for media). The only case where it can elect not to are multicast sessions, where participation is possible even if the other participant in the dialog has terminated its involvement in the session. Whether or not it ends its participation on the session, the UAS core MUST generate a 2xx response to the BYE, and MUST pass that to the server transaction for transmission.

The UAS MUST still respond to any pending requests received for that dialog. It is RECOMMENDED that a 487 (Request Terminated) response be generated to those pending requests.

16 Proxy Behavior

16.1 Overview

SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients. A request may traverse several proxies on its way to a UAS. Each will make routing decisions, modifying the request before forwarding it to the next element. Responses will route through the same set of proxies traversed by the request in the reverse order.

Being a proxy is a logical role for a SIP element. When a request arrives, an element that can play the role of a proxy first decides if it needs to respond to the request on its own. For instance, the request may be malformed or the element may need credentials from the client before acting as a proxy. The element MAY respond with any

appropriate error code. When responding directly to a request, the element is playing the role of a UAS and MUST behave as described in Section 8.2.

A proxy can operate in either a stateful or stateless mode for each new request. When stateless, a proxy acts as a simple forwarding element. It forwards each request downstream to a single element determined by making a targeting and routing decision based on the request. It simply forwards every response it receives upstream. A stateless proxy discards information about a message once the message has been forwarded. A stateful proxy remembers information (specifically, transaction state) about each incoming request and any requests it sends as a result of processing the incoming request. It uses this information to affect the processing of future messages associated with that request. A stateful proxy MAY choose to "fork" a request, routing it to multiple destinations. Any request that is forwarded to more than one location MUST be handled statefully.

In some circumstances, a proxy MAY forward requests using stateful transports (such as TCP) without being transaction-stateful. For instance, a proxy MAY forward a request from one TCP connection to another transaction statelessly as long as it places enough information in the message to be able to forward the response down the same connection the request arrived on. Requests forwarded between different types of transports where the proxy's TU must take an active role in ensuring reliable delivery on one of the transports MUST be forwarded transaction statefully.

A stateful proxy MAY transition to stateless operation at any time during the processing of a request, so long as it did not do anything that would otherwise prevent it from being stateless initially (forking, for example, or generation of a 100 response). When performing such a transition, all state is simply discarded. The proxy SHOULD NOT initiate a CANCEL request.

Much of the processing involved when acting statelessly or statefully for a request is identical. The next several subsections are written from the point of view of a stateful proxy. The last section calls out those places where a stateless proxy behaves differently.

16.2 Stateful Proxy

When stateful, a proxy is purely a SIP transaction processing engine. Its behavior is modeled here in terms of the server and client transactions defined in Section 17. A stateful proxy has a server transaction associated with one or more client transactions by a higher layer proxy processing component (see figure 3), known as a proxy core. An incoming request is processed by a server

transaction. Requests from the server transaction are passed to a proxy core. The proxy core determines where to route the request, choosing one or more next-hop locations. An outgoing request for each next-hop location is processed by its own associated client transaction. The proxy core collects the responses from the client transactions and uses them to send responses to the server transaction.

A stateful proxy creates a new server transaction for each new request received. Any retransmissions of the request will then be handled by that server transaction per Section 17. The proxy core MUST behave as a UAS with respect to sending an immediate provisional on that server transaction (such as 100 Trying) as described in Section 8.2.6. Thus, a stateful proxy SHOULD NOT generate 100 (Trying) responses to non-INVITE requests.

This is a model of proxy behavior, not of software. An implementation is free to take any approach that replicates the external behavior this model defines.

For all new requests, including any with unknown methods, an element intending to proxy the request MUST:

1. Validate the request (Section 16.3)

2. Preprocess routing information (Section 16.4)

3. Determine target(s) for the request (Section 16.5)

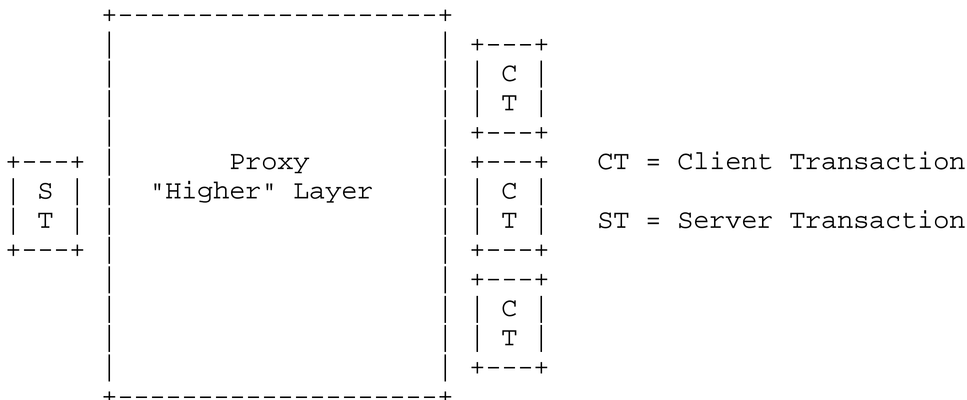


Figure 3: Stateful Proxy Model

4. Forward the request to each target (Section 16.6)

5. Process all responses (Section 16.7)

16.3 Request Validation

Before an element can proxy a request, it MUST verify the message's validity. A valid message must pass the following checks:

1. Reasonable Syntax

2. URI scheme

3. Max-Forwards

4. (Optional) Loop Detection

5. Proxy-Require

6. Proxy-Authorization

If any of these checks fail, the element MUST behave as a user agent server (see Section 8.2) and respond with an error code.

Notice that a proxy is not required to detect merged requests and MUST NOT treat merged requests as an error condition. The endpoints receiving the requests will resolve the merge as described in Section 8.2.2.2.

1. Reasonable syntax check

   The request MUST be well-formed enough to be handled with a server transaction. Any components involved in the remainder of these Request Validation steps or the Request Forwarding section MUST be well-formed. Any other components, well-formed or not, SHOULD be ignored and remain unchanged when the message is forwarded. For instance, an element would not reject a request because of a malformed Date header field. Likewise, a proxy would not remove a malformed Date header field before forwarding a request.

   This protocol is designed to be extended. Future extensions may define new methods and header fields at any time. An element MUST NOT refuse to proxy a request because it contains a method or header field it does not know about.

2. URI scheme check

   If the Request-URI has a URI whose scheme is not understood by the proxy, the proxy SHOULD reject the request with a 416 (Unsupported URI Scheme) response.

3. Max-Forwards check

   The Max-Forwards header field (Section 20.22) is used to limit the number of elements a SIP request can traverse.

   If the request does not contain a Max-Forwards header field, this check is passed.

   If the request contains a Max-Forwards header field with a field value greater than zero, the check is passed.

   If the request contains a Max-Forwards header field with a field value of zero (0), the element MUST NOT forward the request. If the request was for OPTIONS, the element MAY act as the final recipient and respond per Section 11. Otherwise, the element MUST return a 483 (Too many hops) response.

4. Optional Loop Detection check

   An element MAY check for forwarding loops before forwarding a request. If the request contains a Via header field with a sent-by value that equals a value placed into previous requests by the proxy, the request has been forwarded by this element before. The request has either looped or is legitimately spiraling through the element. To determine if the request has looped, the element MAY perform the branch parameter calculation described in Step 8 of Section 16.6 on this message and compare it to the parameter received in that Via header field. If the parameters match, the request has looped. If they differ, the request is spiraling, and processing continues. If a loop is detected, the element MAY return a 482 (Loop Detected) response.

5. Proxy-Require check

   Future extensions to this protocol may introduce features that require special handling by proxies. Endpoints will include a Proxy-Require header field in requests that use these features, telling the proxy not to process the request unless the feature is understood.

If the request contains a Proxy-Require header field (Section 20.29) with one or more option-tags this element does not understand, the element MUST return a 420 (Bad Extension) response. The response MUST include an Unsupported (Section 20.40) header field listing those option-tags the element did not understand.

6. Proxy-Authorization check

If an element requires credentials before forwarding a request, the request MUST be inspected as described in Section 22.3. That section also defines what the element must do if the inspection fails.

16.4 Route Information Preprocessing

The proxy MUST inspect the Request-URI of the request. If the Request-URI of the request contains a value this proxy previously placed into a Record-Route header field (see Section 16.6 item 4), the proxy MUST replace the Request-URI in the request with the last value from the Route header field, and remove that value from the Route header field. The proxy MUST then proceed as if it received this modified request.

This will only happen when the element sending the request to the proxy (which may have been an endpoint) is a strict router. This rewrite on receive is necessary to enable backwards compatibility with those elements. It also allows elements following this specification to preserve the Request-URI through strict-routing proxies (see Section 12.2.1.1).

This requirement does not obligate a proxy to keep state in order to detect URIs it previously placed in Record-Route header fields. Instead, a proxy need only place enough information in those URIs to recognize them as values it provided when they later appear.

If the Request-URI contains a maddr parameter, the proxy MUST check to see if its value is in the set of addresses or domains the proxy is configured to be responsible for. If the Request-URI has a maddr parameter with a value the proxy is responsible for, and the request was received using the port and transport indicated (explicitly or by default) in the Request-URI, the proxy MUST strip the maddr and any non-default port or transport parameter and continue processing as if those values had not been present in the request.

A request may arrive with a maddr matching the proxy, but on a port or transport different from that indicated in the URI. Such a request needs to be forwarded to the proxy using the indicated port and transport.

If the first value in the Route header field indicates this proxy, the proxy MUST remove that value from the request.

16.5 Determining Request Targets

Next, the proxy calculates the target(s) of the request. The set of targets will either be predetermined by the contents of the request or will be obtained from an abstract location service. Each target in the set is represented as a URI.

If the Request-URI of the request contains an maddr parameter, the Request-URI MUST be placed into the target set as the only target URI, and the proxy MUST proceed to Section 16.6.

If the domain of the Request-URI indicates a domain this element is not responsible for, the Request-URI MUST be placed into the target set as the only target, and the element MUST proceed to the task of Request Forwarding (Section 16.6).

There are many circumstances in which a proxy might receive a request for a domain it is not responsible for. A firewall proxy handling outgoing calls (the way HTTP proxies handle outgoing requests) is an example of where this is likely to occur.

If the target set for the request has not been predetermined as described above, this implies that the element is responsible for the domain in the Request-URI, and the element MAY use whatever mechanism it desires to determine where to send the request. Any of these mechanisms can be modeled as accessing an abstract Location Service. This may consist of obtaining information from a location service created by a SIP Registrar, reading a database, consulting a presence server, utilizing other protocols, or simply performing an algorithmic substitution on the Request-URI. When accessing the location service constructed by a registrar, the Request-URI MUST first be canonicalized as described in Section 10.3 before being used as an index. The output of these mechanisms is used to construct the target set.

If the Request-URI does not provide sufficient information for the proxy to determine the target set, it SHOULD return a 485 (Ambiguous) response. This response SHOULD contain a Contact header field containing URIs of new addresses to be tried. For example, an INVITE

to sip:John.Smith@company.com may be ambiguous at a proxy whose location service has multiple John Smiths listed. See Section 21.4.23 for details.

Any information in or about the request or the current environment of the element MAY be used in the construction of the target set. For instance, different sets may be constructed depending on contents or the presence of header fields and bodies, the time of day of the request's arrival, the interface on which the request arrived, failure of previous requests, or even the element's current level of utilization.

As potential targets are located through these services, their URIs are added to the target set. Targets can only be placed in the target set once. If a target URI is already present in the set (based on the definition of equality for the URI type), it MUST NOT be added again.

A proxy MUST NOT add additional targets to the target set if the Request-URI of the original request does not indicate a resource this proxy is responsible for.

> A proxy can only change the Request-URI of a request during forwarding if it is responsible for that URI. If the proxy is not responsible for that URI, it will not recurse on 3xx or 416 responses as described below.

If the Request-URI of the original request indicates a resource this proxy is responsible for, the proxy MAY continue to add targets to the set after beginning Request Forwarding. It MAY use any information obtained during that processing to determine new targets. For instance, a proxy may choose to incorporate contacts obtained in a redirect response (3xx) into the target set. If a proxy uses a dynamic source of information while building the target set (for instance, if it consults a SIP Registrar), it SHOULD monitor that source for the duration of processing the request. New locations SHOULD be added to the target set as they become available. As above, any given URI MUST NOT be added to the set more than once.

> Allowing a URI to be added to the set only once reduces unnecessary network traffic, and in the case of incorporating contacts from redirect requests prevents infinite recursion.

For example, a trivial location service is a "no-op", where the target URI is equal to the incoming request URI. The request is sent to a specific next hop proxy for further processing. During request

forwarding of Section 16.6, Item 6, the identity of that next hop, expressed as a SIP or SIPS URI, is inserted as the top-most Route header field value into the request.

If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404 (Not Found) response.

If the target set remains empty after applying all of the above, the proxy MUST return an error response, which SHOULD be the 480 (Temporarily Unavailable) response.

16.6 Request Forwarding

As soon as the target set is non-empty, a proxy MAY begin forwarding the request. A stateful proxy MAY process the set in any order. It MAY process multiple targets serially, allowing each client transaction to complete before starting the next. It MAY start client transactions with every target in parallel. It also MAY arbitrarily divide the set into groups, processing the groups serially and processing the targets in each group in parallel.

A common ordering mechanism is to use the qvalue parameter of targets obtained from Contact header fields (see Section 20.10). Targets are processed from highest qvalue to lowest. Targets with equal qvalues may be processed in parallel.

A stateful proxy must have a mechanism to maintain the target set as responses are received and associate the responses to each forwarded request with the original request. For the purposes of this model, this mechanism is a "response context" created by the proxy layer before forwarding the first request.

For each target, the proxy forwards the request following these steps:

1. Make a copy of the received request

2. Update the Request-URI

3. Update the Max-Forwards header field

4. Optionally add a Record-route header field value

5. Optionally add additional header fields

6. Postprocess routing information

7. Determine the next-hop address, port, and transport

8. Add a Via header field value

9. Add a Content-Length header field if necessary

10. Forward the new request

11. Set timer C

Each of these steps is detailed below:

1. Copy request

   The proxy starts with a copy of the received request. The copy MUST initially contain all of the header fields from the received request. Fields not detailed in the processing described below MUST NOT be removed. The copy SHOULD maintain the ordering of the header fields as in the received request. The proxy MUST NOT reorder field values with a common field name (See Section 7.3.1). The proxy MUST NOT add to, modify, or remove the message body.

   An actual implementation need not perform a copy; the primary requirement is that the processing for each next hop begin with the same request.

2. Request-URI

   The Request-URI in the copy's start line MUST be replaced with the URI for this target. If the URI contains any parameters not allowed in a Request-URI, they MUST be removed.

   This is the essence of a proxy's role. This is the mechanism through which a proxy routes a request toward its destination.

   In some circumstances, the received Request-URI is placed into the target set without being modified. For that target, the replacement above is effectively a no-op.

3. Max-Forwards

   If the copy contains a Max-Forwards header field, the proxy MUST decrement its value by one (1).

   If the copy does not contain a Max-Forwards header field, the proxy MUST add one with a field value, which SHOULD be 70.

   Some existing UAs will not provide a Max-Forwards header field in a request.

4. Record-Route

If this proxy wishes to remain on the path of future requests in a dialog created by this request (assuming the request creates a dialog), it MUST insert a Record-Route header field value into the copy before any existing Record-Route header field values, even if a Route header field is already present.

Requests establishing a dialog may contain a preloaded Route header field.

If this request is already part of a dialog, the proxy SHOULD insert a Record-Route header field value if it wishes to remain on the path of future requests in the dialog. In normal endpoint operation as described in Section 12, these Record-Route header field values will not have any effect on the route sets used by the endpoints.

The proxy will remain on the path if it chooses to not insert a Record-Route header field value into requests that are already part of a dialog. However, it would be removed from the path when an endpoint that has failed reconstitutes the dialog.

A proxy MAY insert a Record-Route header field value into any request. If the request does not initiate a dialog, the endpoints will ignore the value. See Section 12 for details on how endpoints use the Record-Route header field values to construct Route header fields.

Each proxy in the path of a request chooses whether to add a Record-Route header field value independently - the presence of a Record-Route header field in a request does not obligate this proxy to add a value.

The URI placed in the Record-Route header field value MUST be a SIP or SIPS URI. This URI MUST contain an lr parameter (see Section 19.1.1). This URI MAY be different for each destination the request is forwarded to. The URI SHOULD NOT contain the transport parameter unless the proxy has knowledge (such as in a private network) that the next downstream element that will be in the path of subsequent requests supports that transport.

The URI this proxy provides will be used by some other element to make a routing decision. This proxy, in general, has no way of knowing the capabilities of that element, so it must restrict itself to the mandatory elements of a SIP implementation: SIP URIs and either the TCP or UDP transports.

The URI placed in the Record-Route header field MUST resolve to the element inserting it (or a suitable stand-in) when the server location procedures of [4] are applied to it, so that subsequent requests reach the same SIP element. If the Request-URI contains a SIPS URI, or the topmost Route header field value (after the post processing of bullet 6) contains a SIPS URI, the URI placed into the Record-Route header field MUST be a SIPS URI. Furthermore, if the request was not received over TLS, the proxy MUST insert a Record-Route header field. In a similar fashion, a proxy that receives a request over TLS, but generates a request without a SIPS URI in the Request-URI or topmost Route header field value (after the post processing of bullet 6), MUST insert a Record-Route header field that is not a SIPS URI.

A proxy at a security perimeter must remain on the perimeter throughout the dialog.

If the URI placed in the Record-Route header field needs to be rewritten when it passes back through in a response, the URI MUST be distinct enough to locate at that time. (The request may spiral through this proxy, resulting in more than one Record-Route header field value being added). Item 8 of Section 16.7 recommends a mechanism to make the URI sufficiently distinct.

The proxy MAY include parameters in the Record-Route header field value. These will be echoed in some responses to the request such as the 200 (OK) responses to INVITE. Such parameters may be useful for keeping state in the message rather than the proxy.

If a proxy needs to be in the path of any type of dialog (such as one straddling a firewall), it SHOULD add a Record-Route header field value to every request with a method it does not understand since that method may have dialog semantics.

The URI a proxy places into a Record-Route header field is only valid for the lifetime of any dialog created by the transaction in which it occurs. A dialog-stateful proxy, for example, MAY refuse to accept future requests with that value in the Request-URI after the dialog has terminated. Non-dialog-stateful proxies, of course, have no concept of when the dialog has terminated, but they MAY encode enough information in the value to compare it against the dialog identifier of future requests and MAY reject requests not matching that information. Endpoints MUST NOT use a URI obtained from a Record-Route header field outside the dialog in which it was provided. See

Section 12 for more information on an endpoint's use of Record-Route header fields.

Record-routing may be required by certain services where the proxy needs to observe all messages in a dialog. However, it slows down processing and impairs scalability and thus proxies should only record-route if required for a particular service.

The Record-Route process is designed to work for any SIP request that initiates a dialog. INVITE is the only such request in this specification, but extensions to the protocol MAY define others.

5. Add Additional Header Fields

The proxy MAY add any other appropriate header fields to the copy at this point.

6. Postprocess routing information

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, this can only be known with certainty if the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field of the copy ahead of any existing values, if present. If the Route header field is absent, it MUST be added, containing that list of URIs.

If the proxy has a local policy that mandates that the request visit one specific proxy, an alternative to pushing a Route value into the Route header field is to bypass the forwarding logic of item 10 below, and instead just send the request to the address, port, and transport for that specific proxy. If the request has a Route header field, this alternative MUST NOT be used unless it is known that next hop proxy is a loose router. Otherwise, this approach MAY be used, but the Route insertion mechanism above is preferred for its robustness, flexibility, generality and consistency of operation. Furthermore, if the Request-URI contains a SIPS URI, TLS MUST be used to communicate with that proxy.

If the copy contains a Route header field, the proxy MUST inspect the URI in its first value. If that URI does not contain an lr parameter, the proxy MUST modify the copy as follows:

- The proxy MUST place the Request-URI into the Route header field as the last value.

- The proxy MUST then place the first Route header field value into the Request-URI and remove that value from the Route header field.

Appending the Request-URI to the Route header field is part of a mechanism used to pass the information in that Request-URI through strict-routing elements. "Popping" the first Route header field value into the Request-URI formats the message the way a strict-routing element expects to receive it (with its own URI in the Request-URI and the next location to visit in the first Route header field value).

7. Determine Next-Hop Address, Port, and Transport

The proxy MAY have a local policy to send the request to a specific IP address, port, and transport, independent of the values of the Route and Request-URI. Such a policy MUST NOT be used if the proxy is not certain that the IP address, port, and transport correspond to a server that is a loose router. However, this mechanism for sending the request through a specific next hop is NOT RECOMMENDED; instead a Route header field should be used for that purpose as described above.

In the absence of such an overriding mechanism, the proxy applies the procedures listed in [4] as follows to determine where to send the request. If the proxy has reformatted the request to send to a strict-routing element as described in step 6 above, the proxy MUST apply those procedures to the Request-URI of the request. Otherwise, the proxy MUST apply the procedures to the first value in the Route header field, if present, else the Request-URI. The procedures will produce an ordered set of (address, port, transport) tuples. Independently of which URI is being used as input to the procedures of [4], if the Request-URI specifies a SIPS resource, the proxy MUST follow the procedures of [4] as if the input URI were a SIPS URI.

As described in [4], the proxy MUST attempt to deliver the message to the first tuple in that set, and proceed through the set in order until the delivery attempt succeeds.

For each tuple attempted, the proxy MUST format the message as appropriate for the tuple and send the request using a new client transaction as detailed in steps 8 through 10.

Since each attempt uses a new client transaction, it represents a new branch. Thus, the branch parameter provided with the Via header field inserted in step 8 MUST be different for each attempt.

If the client transaction reports failure to send the request or a timeout from its state machine, the proxy continues to the next address in that ordered set. If the ordered set is exhausted, the request cannot be forwarded to this element in the target set. The proxy does not need to place anything in the response context, but otherwise acts as if this element of the target set returned a 408 (Request Timeout) final response.

8. Add a Via header field value

The proxy MUST insert a Via header field value into the copy before the existing Via header field values. The construction of this value follows the same guidelines of Section 8.1.1.7. This implies that the proxy will compute its own branch parameter, which will be globally unique for that branch, and contain the requisite magic cookie. Note that this implies that the branch parameter will be different for different instances of a spiraled or looped request through a proxy.

Proxies choosing to detect loops have an additional constraint in the value they use for construction of the branch parameter. A proxy choosing to detect loops SHOULD create a branch parameter separable into two parts by the implementation. The first part MUST satisfy the constraints of Section 8.1.1.7 as described above. The second is used to perform loop detection and distinguish loops from spirals.

Loop detection is performed by verifying that, when a request returns to a proxy, those fields having an impact on the processing of the request have not changed. The value placed in this part of the branch parameter SHOULD reflect all of those fields (including any Route, Proxy-Require and Proxy-Authorization header fields). This is to ensure that if the request is routed back to the proxy and one of those fields changes, it is treated as a spiral and not a loop (see Section 16.3). A common way to create this value is to compute a cryptographic hash of the To tag, From tag, Call-ID header field, the Request-URI of the request received (before translation), the topmost Via header, and the sequence number from the CSeq header field, in addition to any Proxy-Require and Proxy-Authorization header fields that may be present. The

algorithm used to compute the hash is implementation-dependent, but MD5 (RFC 1321 [35]), expressed in hexadecimal, is a reasonable choice. (Base64 is not permissible for a token.)

If a proxy wishes to detect loops, the "branch" parameter it supplies MUST depend on all information affecting processing of a request, including the incoming Request-URI and any header fields affecting the request's admission or routing. This is necessary to distinguish looped requests from requests whose routing parameters have changed before returning to this server.

The request method MUST NOT be included in the calculation of the branch parameter. In particular, CANCEL and ACK requests (for non-2xx responses) MUST have the same branch value as the corresponding request they cancel or acknowledge. The branch parameter is used in correlating those requests at the server handling them (see Sections 17.2.3 and 9.2).

9. Add a Content-Length header field if necessary

   If the request will be sent to the next hop using a stream-based transport and the copy contains no Content-Length header field, the proxy MUST insert one with the correct value for the body of the request (see Section 20.14).

10. Forward Request

   A stateful proxy MUST create a new client transaction for this request as described in Section 17.1 and instructs the transaction to send the request using the address, port and transport determined in step 7.

11. Set timer C

   In order to handle the case where an INVITE request never generates a final response, the TU uses a timer which is called timer C. Timer C MUST be set for each client transaction when an INVITE request is proxied. The timer MUST be larger than 3 minutes. Section 16.7 bullet 2 discusses how this timer is updated with provisional responses, and Section 16.8 discusses processing when it fires.

16.7 Response Processing

When a response is received by an element, it first tries to locate a client transaction (Section 17.1.3) matching the response. If none is found, the element MUST process the response (even if it is an informational response) as a stateless proxy (described below). If a match is found, the response is handed to the client transaction.

Forwarding responses for which a client transaction (or more generally any knowledge of having sent an associated request) is not found improves robustness. In particular, it ensures that "late" 2xx responses to INVITE requests are forwarded properly.

As client transactions pass responses to the proxy layer, the following processing MUST take place:

1. Find the appropriate response context

2. Update timer C for provisional responses

3. Remove the topmost Via

4. Add the response to the response context

5. Check to see if this response should be forwarded immediately

6. When necessary, choose the best final response from the response context

If no final response has been forwarded after every client transaction associated with the response context has been terminated, the proxy must choose and forward the "best" response from those it has seen so far.

The following processing MUST be performed on each response that is forwarded. It is likely that more than one response to each request will be forwarded: at least each provisional and one final response.

7. Aggregate authorization header field values if necessary

8. Optionally rewrite Record-Route header field values

9. Forward the response

10. Generate any necessary CANCEL requests

Each of the above steps are detailed below:

1. Find Context

   The proxy locates the "response context" it created before forwarding the original request using the key described in Section 16.6. The remaining processing steps take place in this context.

2. Update timer C for provisional responses

   For an INVITE transaction, if the response is a provisional response with status codes 101 to 199 inclusive (i.e., anything but 100), the proxy MUST reset timer C for that client transaction. The timer MAY be reset to a different value, but this value MUST be greater than 3 minutes.

3. Via

   The proxy removes the topmost Via header field value from the response.

   If no Via header field values remain in the response, the response was meant for this element and MUST NOT be forwarded. The remainder of the processing described in this section is not performed on this message, the UAC processing rules described in Section 8.1.3 are followed instead (transport layer processing has already occurred).

   This will happen, for instance, when the element generates CANCEL requests as described in Section 10.

4. Add response to context

   Final responses received are stored in the response context until a final response is generated on the server transaction associated with this context. The response may be a candidate for the best final response to be returned on that server transaction. Information from this response may be needed in forming the best response, even if this response is not chosen.

   If the proxy chooses to recurse on any contacts in a 3xx response by adding them to the target set, it MUST remove them from the response before adding the response to the response context. However, a proxy SHOULD NOT recurse to a non-SIPS URI if the Request-URI of the original request was a SIPS URI. If

the proxy recurses on all of the contacts in a 3xx response, the proxy SHOULD NOT add the resulting contactless response to the response context.

Removing the contact before adding the response to the response context prevents the next element upstream from retrying a location this proxy has already attempted.

3xx responses may contain a mixture of SIP, SIPS, and non-SIP URIs. A proxy may choose to recurse on the SIP and SIPS URIs and place the remainder into the response context to be returned, potentially in the final response.

If a proxy receives a 416 (Unsupported URI Scheme) response to a request whose Request-URI scheme was not SIP, but the scheme in the original received request was SIP or SIPS (that is, the proxy changed the scheme from SIP or SIPS to something else when it proxied a request), the proxy SHOULD add a new URI to the target set. This URI SHOULD be a SIP URI version of the non-SIP URI that was just tried. In the case of the tel URL, this is accomplished by placing the telephone-subscriber part of the tel URL into the user part of the SIP URI, and setting the hostpart to the domain where the prior request was sent. See Section 19.1.6 for more detail on forming SIP URIs from tel URLs.

As with a 3xx response, if a proxy "recurses" on the 416 by trying a SIP or SIPS URI instead, the 416 response SHOULD NOT be added to the response context.

5. Check response for forwarding

Until a final response has been sent on the server transaction, the following responses MUST be forwarded immediately:

- Any provisional response other than 100 (Trying)

- Any 2xx response

If a 6xx response is received, it is not immediately forwarded, but the stateful proxy SHOULD cancel all client pending transactions as described in Section 10, and it MUST NOT create any new branches in this context.

This is a change from RFC 2543, which mandated that the proxy was to forward the 6xx response immediately. For an INVITE transaction, this approach had the problem that a 2xx response could arrive on another branch, in which case the proxy would

have to forward the 2xx. The result was that the UAC could receive a 6xx response followed by a 2xx response, which should never be allowed to happen. Under the new rules, upon receiving a 6xx, a proxy will issue a CANCEL request, which will generally result in 487 responses from all outstanding client transactions, and then at that point the 6xx is forwarded upstream.

After a final response has been sent on the server transaction, the following responses MUST be forwarded immediately:

- Any 2xx response to an INVITE request

A stateful proxy MUST NOT immediately forward any other responses. In particular, a stateful proxy MUST NOT forward any 100 (Trying) response. Those responses that are candidates for forwarding later as the "best" response have been gathered as described in step "Add Response to Context".

Any response chosen for immediate forwarding MUST be processed as described in steps "Aggregate Authorization Header Field Values" through "Record-Route".

This step, combined with the next, ensures that a stateful proxy will forward exactly one final response to a non-INVITE request, and either exactly one non-2xx response or one or more 2xx responses to an INVITE request.

6. Choosing the best response

A stateful proxy MUST send a final response to a response context's server transaction if no final responses have been immediately forwarded by the above rules and all client transactions in this response context have been terminated.

The stateful proxy MUST choose the "best" final response among those received and stored in the response context.

If there are no final responses in the context, the proxy MUST send a 408 (Request Timeout) response to the server transaction.

Otherwise, the proxy MUST forward a response from the responses stored in the response context. It MUST choose from the 6xx class responses if any exist in the context. If no 6xx class responses are present, the proxy SHOULD choose from the lowest response class stored in the response context. The proxy MAY select any response within that chosen class. The proxy SHOULD

give preference to responses that provide information affecting resubmission of this request, such as 401, 407, 415, 420, and 484 if the 4xx class is chosen.

A proxy which receives a 503 (Service Unavailable) response SHOULD NOT forward it upstream unless it can determine that any subsequent requests it might proxy will also generate a 503. In other words, forwarding a 503 means that the proxy knows it cannot service any requests, not just the one for the Request-URI in the request which generated the 503. If the only response that was received is a 503, the proxy SHOULD generate a 500 response and forward that upstream.

The forwarded response MUST be processed as described in steps "Aggregate Authorization Header Field Values" through "Record-Route".

For example, if a proxy forwarded a request to 4 locations, and received 503, 407, 501, and 404 responses, it may choose to forward the 407 (Proxy Authentication Required) response.

1xx and 2xx responses may be involved in the establishment of dialogs. When a request does not contain a To tag, the To tag in the response is used by the UAC to distinguish multiple responses to a dialog creating request. A proxy MUST NOT insert a tag into the To header field of a 1xx or 2xx response if the request did not contain one. A proxy MUST NOT modify the tag in the To header field of a 1xx or 2xx response.

Since a proxy may not insert a tag into the To header field of a 1xx response to a request that did not contain one, it cannot issue non-100 provisional responses on its own. However, it can branch the request to a UAS sharing the same element as the proxy. This UAS can return its own provisional responses, entering into an early dialog with the initiator of the request. The UAS does not have to be a discreet process from the proxy. It could be a virtual UAS implemented in the same code space as the proxy.

3-6xx responses are delivered hop-by-hop. When issuing a 3-6xx response, the element is effectively acting as a UAS, issuing its own response, usually based on the responses received from downstream elements. An element SHOULD preserve the To tag when simply forwarding a 3-6xx response to a request that did not contain a To tag.

A proxy MUST NOT modify the To tag in any forwarded response to a request that contains a To tag.

While it makes no difference to the upstream elements if the proxy replaced the To tag in a forwarded 3-6xx response, preserving the original tag may assist with debugging.

When the proxy is aggregating information from several responses, choosing a To tag from among them is arbitrary, and generating a new To tag may make debugging easier. This happens, for instance, when combining 401 (Unauthorized) and 407 (Proxy Authentication Required) challenges, or combining Contact values from unencrypted and unauthenticated 3xx responses.

7. Aggregate Authorization Header Field Values

If the selected response is a 401 (Unauthorized) or 407 (Proxy Authentication Required), the proxy MUST collect any WWW-Authenticate and Proxy-Authenticate header field values from all other 401 (Unauthorized) and 407 (Proxy Authentication Required) responses received so far in this response context and add them to this response without modification before forwarding. The resulting 401 (Unauthorized) or 407 (Proxy Authentication Required) response could have several WWW-Authenticate AND Proxy-Authenticate header field values.

This is necessary because any or all of the destinations the request was forwarded to may have requested credentials. The client needs to receive all of those challenges and supply credentials for each of them when it retries the request. Motivation for this behavior is provided in Section 26.

8. Record-Route

If the selected response contains a Record-Route header field value originally provided by this proxy, the proxy MAY choose to rewrite the value before forwarding the response. This allows the proxy to provide different URIs for itself to the next upstream and downstream elements. A proxy may choose to use this mechanism for any reason. For instance, it is useful for multi-homed hosts.

If the proxy received the request over TLS, and sent it out over a non-TLS connection, the proxy MUST rewrite the URI in the Record-Route header field to be a SIPS URI. If the proxy received the request over a non-TLS connection, and sent it out over TLS, the proxy MUST rewrite the URI in the Record-Route header field to be a SIP URI.

The new URI provided by the proxy MUST satisfy the same constraints on URIs placed in Record-Route header fields in requests (see Step 4 of Section 16.6) with the following modifications:

The URI SHOULD NOT contain the transport parameter unless the proxy has knowledge that the next upstream (as opposed to downstream) element that will be in the path of subsequent requests supports that transport.

When a proxy does decide to modify the Record-Route header field in the response, one of the operations it performs is locating the Record-Route value that it had inserted. If the request spiraled, and the proxy inserted a Record-Route value in each iteration of the spiral, locating the correct value in the response (which must be the proper iteration in the reverse direction) is tricky. The rules above recommend that a proxy wishing to rewrite Record-Route header field values insert sufficiently distinct URIs into the Record-Route header field so that the right one may be selected for rewriting. A RECOMMENDED mechanism to achieve this is for the proxy to append a unique identifier for the proxy instance to the user portion of the URI.

When the response arrives, the proxy modifies the first Record-Route whose identifier matches the proxy instance. The modification results in a URI without this piece of data appended to the user portion of the URI. Upon the next iteration, the same algorithm (find the topmost Record-Route header field value with the parameter) will correctly extract the next Record-Route header field value inserted by that proxy.

Not every response to a request to which a proxy adds a Record-Route header field value will contain a Record-Route header field. If the response does contain a Record-Route header field, it will contain the value the proxy added.

9. Forward response

After performing the processing described in steps "Aggregate Authorization Header Field Values" through "Record-Route", the proxy MAY perform any feature specific manipulations on the selected response. The proxy MUST NOT add to, modify, or remove the message body. Unless otherwise specified, the proxy MUST NOT remove any header field values other than the Via header field value discussed in Section 16.7 Item 3. In particular, the proxy MUST NOT remove any "received" parameter

it may have added to the next Via header field value while processing the request associated with this response. The proxy MUST pass the response to the server transaction associated with the response context. This will result in the response being sent to the location now indicated in the topmost Via header field value. If the server transaction is no longer available to handle the transmission, the element MUST forward the response statelessly by sending it to the server transport. The server transaction might indicate failure to send the response or signal a timeout in its state machine. These errors would be logged for diagnostic purposes as appropriate, but the protocol requires no remedial action from the proxy.

The proxy MUST maintain the response context until all of its associated transactions have been terminated, even after forwarding a final response.

10. Generate CANCELs

If the forwarded response was a final response, the proxy MUST generate a CANCEL request for all pending client transactions associated with this response context. A proxy SHOULD also generate a CANCEL request for all pending client transactions associated with this response context when it receives a 6xx response. A pending client transaction is one that has received a provisional response, but no final response (it is in the proceeding state) and has not had an associated CANCEL generated for it. Generating CANCEL requests is described in Section 9.1.

The requirement to CANCEL pending client transactions upon forwarding a final response does not guarantee that an endpoint will not receive multiple 200 (OK) responses to an INVITE. 200 (OK) responses on more than one branch may be generated before the CANCEL requests can be sent and processed. Further, it is reasonable to expect that a future extension may override this requirement to issue CANCEL requests.

16.8 Processing Timer C

If timer C should fire, the proxy MUST either reset the timer with any value it chooses, or terminate the client transaction. If the client transaction has received a provisional response, the proxy MUST generate a CANCEL request matching that transaction. If the client transaction has not received a provisional response, the proxy MUST behave as if the transaction received a 408 (Request Timeout) response.

Allowing the proxy to reset the timer allows the proxy to dynamically extend the transaction's lifetime based on current conditions (such as utilization) when the timer fires.

16.9 Handling Transport Errors

If the transport layer notifies a proxy of an error when it tries to forward a request (see Section 18.4), the proxy MUST behave as if the forwarded request received a 503 (Service Unavailable) response.

If the proxy is notified of an error when forwarding a response, it drops the response. The proxy SHOULD NOT cancel any outstanding client transactions associated with this response context due to this notification.

> If a proxy cancels its outstanding client transactions, a single malicious or misbehaving client can cause all transactions to fail through its Via header field.

16.10 CANCEL Processing

A stateful proxy MAY generate a CANCEL to any other request it has generated at any time (subject to receiving a provisional response to that request as described in section 9.1). A proxy MUST cancel any pending client transactions associated with a response context when it receives a matching CANCEL request.

A stateful proxy MAY generate CANCEL requests for pending INVITE client transactions based on the period specified in the INVITE's Expires header field elapsing. However, this is generally unnecessary since the endpoints involved will take care of signaling the end of the transaction.

While a CANCEL request is handled in a stateful proxy by its own server transaction, a new response context is not created for it. Instead, the proxy layer searches its existing response contexts for the server transaction handling the request associated with this CANCEL. If a matching response context is found, the element MUST immediately return a 200 (OK) response to the CANCEL request. In this case, the element is acting as a user agent server as defined in Section 8.2. Furthermore, the element MUST generate CANCEL requests for all pending client transactions in the context as described in Section 16.7 step 10.

If a response context is not found, the element does not have any knowledge of the request to apply the CANCEL to. It MUST statelessly forward the CANCEL request (it may have statelessly forwarded the associated request previously).

16.11 Stateless Proxy

When acting statelessly, a proxy is a simple message forwarder. Much of the processing performed when acting statelessly is the same as when behaving statefully. The differences are detailed here.

A stateless proxy does not have any notion of a transaction, or of the response context used to describe stateful proxy behavior. Instead, the stateless proxy takes messages, both requests and responses, directly from the transport layer (See section 18). As a result, stateless proxies do not retransmit messages on their own. They do, however, forward all retransmissions they receive (they do not have the ability to distinguish a retransmission from the original message). Furthermore, when handling a request statelessly, an element MUST NOT generate its own 100 (Trying) or any other provisional response.

A stateless proxy MUST validate a request as described in Section 16.3

A stateless proxy MUST follow the request processing steps described in Sections 16.4 through 16.5 with the following exception:

- A stateless proxy MUST choose one and only one target from the target set. This choice MUST only rely on fields in the message and time-invariant properties of the server. In particular, a retransmitted request MUST be forwarded to the same destination each time it is processed. Furthermore, CANCEL and non-Routed ACK requests MUST generate the same choice as their associated INVITE.

A stateless proxy MUST follow the request processing steps described in Section 16.6 with the following exceptions:

- The requirement for unique branch IDs across space and time applies to stateless proxies as well. However, a stateless proxy cannot simply use a random number generator to compute the first component of the branch ID, as described in Section 16.6 bullet 8. This is because retransmissions of a request need to have the same value, and a stateless proxy cannot tell a retransmission from the original request. Therefore, the component of the branch parameter that makes it unique MUST be the same each time a retransmitted request is forwarded. Thus for a stateless proxy, the branch parameter MUST be computed as a combinatoric function of message parameters which are invariant on retransmission.

The stateless proxy MAY use any technique it likes to guarantee uniqueness of its branch IDs across transactions. However, the following procedure is RECOMMENDED. The proxy examines the branch ID in the topmost Via header field of the received request. If it begins with the magic cookie, the first component of the branch ID of the outgoing request is computed as a hash of the received branch ID. Otherwise, the first component of the branch ID is computed as a hash of the topmost Via, the tag in the To header field, the tag in the From header field, the Call-ID header field, the CSeq number (but not method), and the Request-URI from the received request. One of these fields will always vary across two different transactions.

- o All other message transformations specified in Section 16.6 MUST result in the same transformation of a retransmitted request. In particular, if the proxy inserts a Record-Route value or pushes URIs into the Route header field, it MUST place the same values in retransmissions of the request. As for the Via branch parameter, this implies that the transformations MUST be based on time-invariant configuration or retransmission-invariant properties of the request.

- o A stateless proxy determines where to forward the request as described for stateful proxies in Section 16.6 Item 10. The request is sent directly to the transport layer instead of through a client transaction.

  Since a stateless proxy must forward retransmitted requests to the same destination and add identical branch parameters to each of them, it can only use information from the message itself and time-invariant configuration data for those calculations. If the configuration state is not time-invariant (for example, if a routing table is updated) any requests that could be affected by the change may not be forwarded statelessly during an interval equal to the transaction timeout window before or after the change. The method of processing the affected requests in that interval is an implementation decision. A common solution is to forward them transaction statefully.

Stateless proxies MUST NOT perform special processing for CANCEL requests. They are processed by the above rules as any other requests. In particular, a stateless proxy applies the same Route header field processing to CANCEL requests that it applies to any other request.

Response processing as described in Section 16.7 does not apply to a proxy behaving statelessly. When a response arrives at a stateless proxy, the proxy MUST inspect the sent-by value in the first (topmost) Via header field value. If that address matches the proxy, (it equals a value this proxy has inserted into previous requests) the proxy MUST remove that header field value from the response and forward the result to the location indicated in the next Via header field value. The proxy MUST NOT add to, modify, or remove the message body. Unless specified otherwise, the proxy MUST NOT remove any other header field values. If the address does not match the proxy, the message MUST be silently discarded.

16.12 Summary of Proxy Route Processing

In the absence of local policy to the contrary, the processing a proxy performs on a request containing a Route header field can be summarized in the following steps.

1. The proxy will inspect the Request-URI. If it indicates a resource owned by this proxy, the proxy will replace it with the results of running a location service. Otherwise, the proxy will not change the Request-URI.

2. The proxy will inspect the URI in the topmost Route header field value. If it indicates this proxy, the proxy removes it from the Route header field (this route node has been reached).

3. The proxy will forward the request to the resource indicated by the URI in the topmost Route header field value or in the Request-URI if no Route header field is present. The proxy determines the address, port and transport to use when forwarding the request by applying the procedures in [4] to that URI.

If no strict-routing elements are encountered on the path of the request, the Request-URI will always indicate the target of the request.

16.12.1 Examples

16.12.1.1 Basic SIP Trapezoid

This scenario is the basic SIP trapezoid, U1 -> P1 -> P2 -> U2, with both proxies record-routing. Here is the flow.

U1 sends:

```
INVITE sip:callee@domain.com SIP/2.0
Contact: sip:caller@u1.example.com
```

to P1. P1 is an outbound proxy. P1 is not responsible for domain.com, so it looks it up in DNS and sends it there. It also adds a Record-Route header field value:

```
INVITE sip:callee@domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p1.example.com;lr>
```

P2 gets this. It is responsible for domain.com so it runs a location service and rewrites the Request-URI. It also adds a Record-Route header field value. There is no Route header field, so it resolves the new Request-URI to determine where to send the request:

```
INVITE sip:callee@u2.domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p2.domain.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

The callee at u2.domain.com gets this and responds with a 200 OK:

```
SIP/2.0 200 OK
Contact: sip:callee@u2.domain.com
Record-Route: <sip:p2.domain.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

The callee at u2 also sets its dialog state’s remote target URI to sip:caller@u1.example.com and its route set to:

```
(<sip:p2.domain.com;lr>,<sip:p1.example.com;lr>)
```

This is forwarded by P2 to P1 to U1 as normal. Now, U1 sets its dialog state’s remote target URI to sip:callee@u2.domain.com and its route set to:

```
(<sip:p1.example.com;lr>,<sip:p2.domain.com;lr>)
```

Since all the route set elements contain the lr parameter, U1 constructs the following BYE request:

```
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>
```

As any other element (including proxies) would do, it resolves the URI in the topmost Route header field value using DNS to determine where to send the request. This goes to P1. P1 notices that it is not responsible for the resource indicated in the Request-URI so it doesn't change it. It does see that it is the first value in the Route header field, so it removes that value, and forwards the request to P2:

```
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p2.domain.com;lr>
```

P2 also notices it is not responsible for the resource indicated by the Request-URI (it is responsible for domain.com, not u2.domain.com), so it doesn't change it. It does see itself in the first Route header field value, so it removes it and forwards the following to u2.domain.com based on a DNS lookup against the Request-URI:

```
BYE sip:callee@u2.domain.com SIP/2.0
```

16.12.1.2 Traversing a Strict-Routing Proxy

In this scenario, a dialog is established across four proxies, each of which adds Record-Route header field values. The third proxy implements the strict-routing procedures specified in RFC 2543 and many works in progress.

```
U1->P1->P2->P3->P4->U2
```

The INVITE arriving at U2 contains:

```
INVITE sip:callee@u2.domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p4.domain.com;lr>
Record-Route: <sip:p3.middle.com>
Record-Route: <sip:p2.example.com;lr>
Record-Route: <sip:p1.example.com;lr>
```

Which U2 responds to with a 200 OK. Later, U2 sends the following BYE request to P4 based on the first Route header field value.

```
BYE sip:caller@u1.example.com SIP/2.0
Route: <sip:p4.domain.com;lr>
Route: <sip:p3.middle.com>
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>
```

P4 is not responsible for the resource indicated in the Request-URI so it will leave it alone. It notices that it is the element in the first Route header field value so it removes it. It then prepares to send the request based on the now first Route header field value of sip:p3.middle.com, but it notices that this URI does not contain the lr parameter, so before sending, it reformats the request to be:

```
BYE sip:p3.middle.com SIP/2.0
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>
```

P3 is a strict router, so it forwards the following to P2:

```
BYE sip:p2.example.com;lr SIP/2.0
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>
```

P2 sees the request-URI is a value it placed into a Record-Route header field, so before further processing, it rewrites the request to be:

```
BYE sip:caller@u1.example.com SIP/2.0
Route: <sip:p1.example.com;lr>
```

P2 is not responsible for u1.example.com, so it sends the request to P1 based on the resolution of the Route header field value.

P1 notices itself in the topmost Route header field value, so it removes it, resulting in:

```
BYE sip:caller@u1.example.com SIP/2.0
```

Since P1 is not responsible for u1.example.com and there is no Route header field, P1 will forward the request to u1.example.com based on the Request-URI.

16.12.1.3 Rewriting Record-Route Header Field Values

In this scenario, U1 and U2 are in different private namespaces and they enter a dialog through a proxy P1, which acts as a gateway between the namespaces.

```
U1->P1->U2
```

U1 sends:

```
INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
```

P1 uses its location service and sends the following to U2:

```
INVITE sip:callee@rightprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
Record-Route: <sip:gateway.rightprivatespace.com;lr>
```

U2 sends this 200 (OK) back to P1:

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.rightprivatespace.com;lr>
```

P1 rewrites its Record-Route header parameter to provide a value that U1 will find useful, and sends the following to U1:

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.leftprivatespace.com;lr>
```

Later, U1 sends the following BYE request to P1:

```
BYE sip:callee@u2.rightprivatespace.com SIP/2.0
Route: <sip:gateway.leftprivatespace.com;lr>
```

which P1 forwards to U2 as:

```
BYE sip:callee@u2.rightprivatespace.com SIP/2.0
```

17 Transactions

SIP is a transactional protocol: interactions between components take place in a series of independent message exchanges. Specifically, a SIP transaction consists of a single request and any responses to that request, which include zero or more provisional responses and one or more final responses. In the case of a transaction where the request was an INVITE (known as an INVITE transaction), the transaction also includes the ACK only if the final response was not a 2xx response. If the response was a 2xx, the ACK is not considered part of the transaction.

The reason for this separation is rooted in the importance of delivering all 200 (OK) responses to an INVITE to the UAC. To deliver them all to the UAC, the UAS alone takes responsibility

for retransmitting them (see Section 13.3.1.4), and the UAC alone takes responsibility for acknowledging them with ACK (see Section 13.2.2.4). Since this ACK is retransmitted only by the UAC, it is effectively considered its own transaction.

Transactions have a client side and a server side. The client side is known as a client transaction and the server side as a server transaction. The client transaction sends the request, and the server transaction sends the response. The client and server transactions are logical functions that are embedded in any number of elements. Specifically, they exist within user agents and stateful proxy servers. Consider the example in Section 4. In this example, the UAC executes the client transaction, and its outbound proxy executes the server transaction. The outbound proxy also executes a client transaction, which sends the request to a server transaction in the inbound proxy. That proxy also executes a client transaction, which in turn sends the request to a server transaction in the UAS. This is shown in Figure 4.

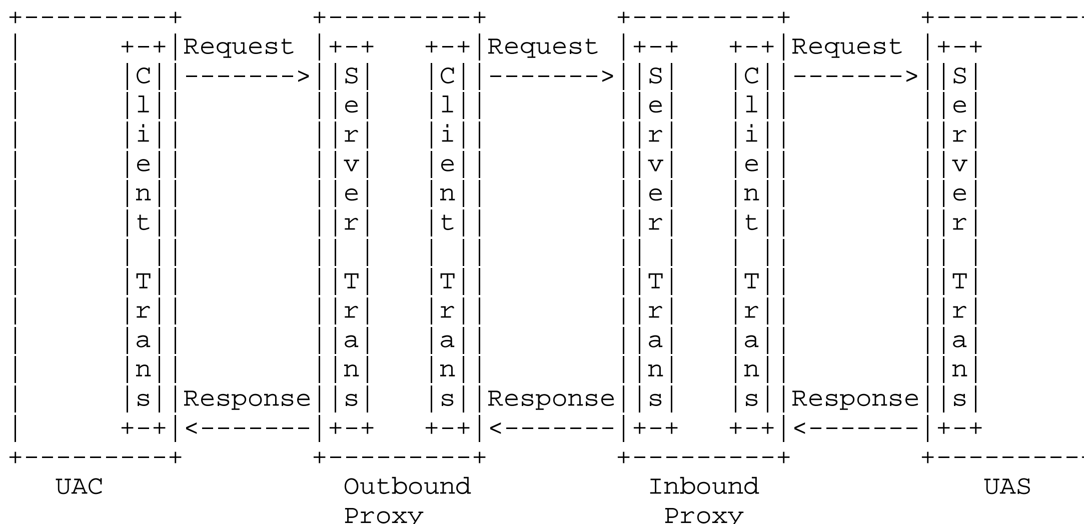


Figure 4: Transaction relationships

A stateless proxy does not contain a client or server transaction. The transaction exists between the UA or stateful proxy on one side, and the UA or stateful proxy on the other side. As far as SIP transactions are concerned, stateless proxies are effectively transparent. The purpose of the client transaction is to receive a request from the element in which the client is embedded (call this element the "Transaction User" or TU; it can be a UA or a stateful proxy), and reliably deliver the request to a server transaction.

The client transaction is also responsible for receiving responses and delivering them to the TU, filtering out any response retransmissions or disallowed responses (such as a response to ACK). Additionally, in the case of an INVITE request, the client transaction is responsible for generating the ACK request for any final response accepting a 2xx response.

Similarly, the purpose of the server transaction is to receive requests from the transport layer and deliver them to the TU. The server transaction filters any request retransmissions from the network. The server transaction accepts responses from the TU and delivers them to the transport layer for transmission over the network. In the case of an INVITE transaction, it absorbs the ACK request for any final response excepting a 2xx response.

The 2xx response and its ACK receive special treatment. This response is retransmitted only by a UAS, and its ACK generated only by the UAC. This end-to-end treatment is needed so that a caller knows the entire set of users that have accepted the call. Because of this special handling, retransmissions of the 2xx response are handled by the UA core, not the transaction layer. Similarly, generation of the ACK for the 2xx is handled by the UA core. Each proxy along the path merely forwards each 2xx response to INVITE and its corresponding ACK.

17.1 Client Transaction

The client transaction provides its functionality through the maintenance of a state machine.

The TU communicates with the client transaction through a simple interface. When the TU wishes to initiate a new transaction, it creates a client transaction and passes it the SIP request to send and an IP address, port, and transport to which to send it. The client transaction begins execution of its state machine. Valid responses are passed up to the TU from the client transaction.

There are two types of client transaction state machines, depending on the method of the request passed by the TU. One handles client transactions for INVITE requests. This type of machine is referred to as an INVITE client transaction. Another type handles client transactions for all requests except INVITE and ACK. This is referred to as a non-INVITE client transaction. There is no client transaction for ACK. If the TU wishes to send an ACK, it passes one directly to the transport layer for transmission.

The INVITE transaction is different from those of other methods because of its extended duration. Normally, human input is required in order to respond to an INVITE. The long delays expected for sending a response argue for a three-way handshake. On the other hand, requests of other methods are expected to complete rapidly. Because of the non-INVITE transaction's reliance on a two-way handshake, TUs SHOULD respond immediately to non-INVITE requests.

17.1.1 INVITE Client Transaction

17.1.1.1 Overview of INVITE Transaction

The INVITE transaction consists of a three-way handshake. The client transaction sends an INVITE, the server transaction sends responses, and the client transaction sends an ACK. For unreliable transports (such as UDP), the client transaction retransmits requests at an interval that starts at T1 seconds and doubles after every retransmission. T1 is an estimate of the round-trip time (RTT), and it defaults to 500 ms. Nearly all of the transaction timers described here scale with T1, and changing T1 adjusts their values. The request is not retransmitted over reliable transports. After receiving a 1xx response, any retransmissions cease altogether, and the client waits for further responses. The server transaction can send additional 1xx responses, which are not transmitted reliably by the server transaction. Eventually, the server transaction decides to send a final response. For unreliable transports, that response is retransmitted periodically, and for reliable transports, it is sent once. For each final response that is received at the client transaction, the client transaction sends an ACK, the purpose of which is to quench retransmissions of the response.

17.1.1.2 Formal Description

The state machine for the INVITE client transaction is shown in Figure 5. The initial state, "calling", MUST be entered when the TU initiates a new client transaction with an INVITE request. The client transaction MUST pass the request to the transport layer for transmission (see Section 18). If an unreliable transport is being used, the client transaction MUST start timer A with a value of T1. If a reliable transport is being used, the client transaction SHOULD NOT start timer A (Timer A controls request retransmissions). For any transport, the client transaction MUST start timer B with a value of 64*T1 seconds (Timer B controls transaction timeouts).

When timer A fires, the client transaction MUST retransmit the request by passing it to the transport layer, and MUST reset the timer with a value of 2*T1. The formal definition of retransmit

within the context of the transaction layer is to take the message previously sent to the transport layer and pass it to the transport layer once more.

When timer A fires 2*T1 seconds later, the request MUST be retransmitted again (assuming the client transaction is still in this state). This process MUST continue so that the request is retransmitted with intervals that double after each transmission. These retransmissions SHOULD only be done while the client transaction is in the "calling" state.

The default value for T1 is 500 ms. T1 is an estimate of the RTT between the client and server transactions. Elements MAY (though it is NOT RECOMMENDED) use smaller values of T1 within closed, private networks that do not permit general Internet connection. T1 MAY be chosen larger, and this is RECOMMENDED if it is known in advance (such as on high latency access links) that the RTT is larger. Whatever the value of T1, the exponential backoffs on retransmissions described in this section MUST be used.

If the client transaction is still in the "Calling" state when timer B fires, the client transaction SHOULD inform the TU that a timeout has occurred. The client transaction MUST NOT generate an ACK. The value of 64*T1 is equal to the amount of time required to send seven requests in the case of an unreliable transport.

If the client transaction receives a provisional response while in the "Calling" state, it transitions to the "Proceeding" state. In the "Proceeding" state, the client transaction SHOULD NOT retransmit the request any longer. Furthermore, the provisional response MUST be passed to the TU. Any further provisional responses MUST be passed up to the TU while in the "Proceeding" state.

When in either the "Calling" or "Proceeding" states, reception of a response with status code from 300-699 MUST cause the client transaction to transition to "Completed". The client transaction MUST pass the received response up to the TU, and the client transaction MUST generate an ACK request, even if the transport is reliable (guidelines for constructing the ACK from the response are given in Section 17.1.1.3) and then pass the ACK to the transport layer for transmission. The ACK MUST be sent to the same address, port, and transport to which the original request was sent. The client transaction SHOULD start timer D when it enters the "Completed" state, with a value of at least 32 seconds for unreliable transports, and a value of zero seconds for reliable transports. Timer D reflects the amount of time that the server transaction can remain in the "Completed" state when unreliable transports are used. This is equal to Timer H in the INVITE server transaction, whose

default is 64*T1. However, the client transaction does not know the value of T1 in use by the server transaction, so an absolute minimum of 32s is used instead of basing Timer D on T1.

Any retransmissions of the final response that are received while in the "Completed" state MUST cause the ACK to be re-passed to the transport layer for retransmission, but the newly received response MUST NOT be passed up to the TU. A retransmission of the response is defined as any response which would match the same client transaction based on the rules of Section 17.1.3.

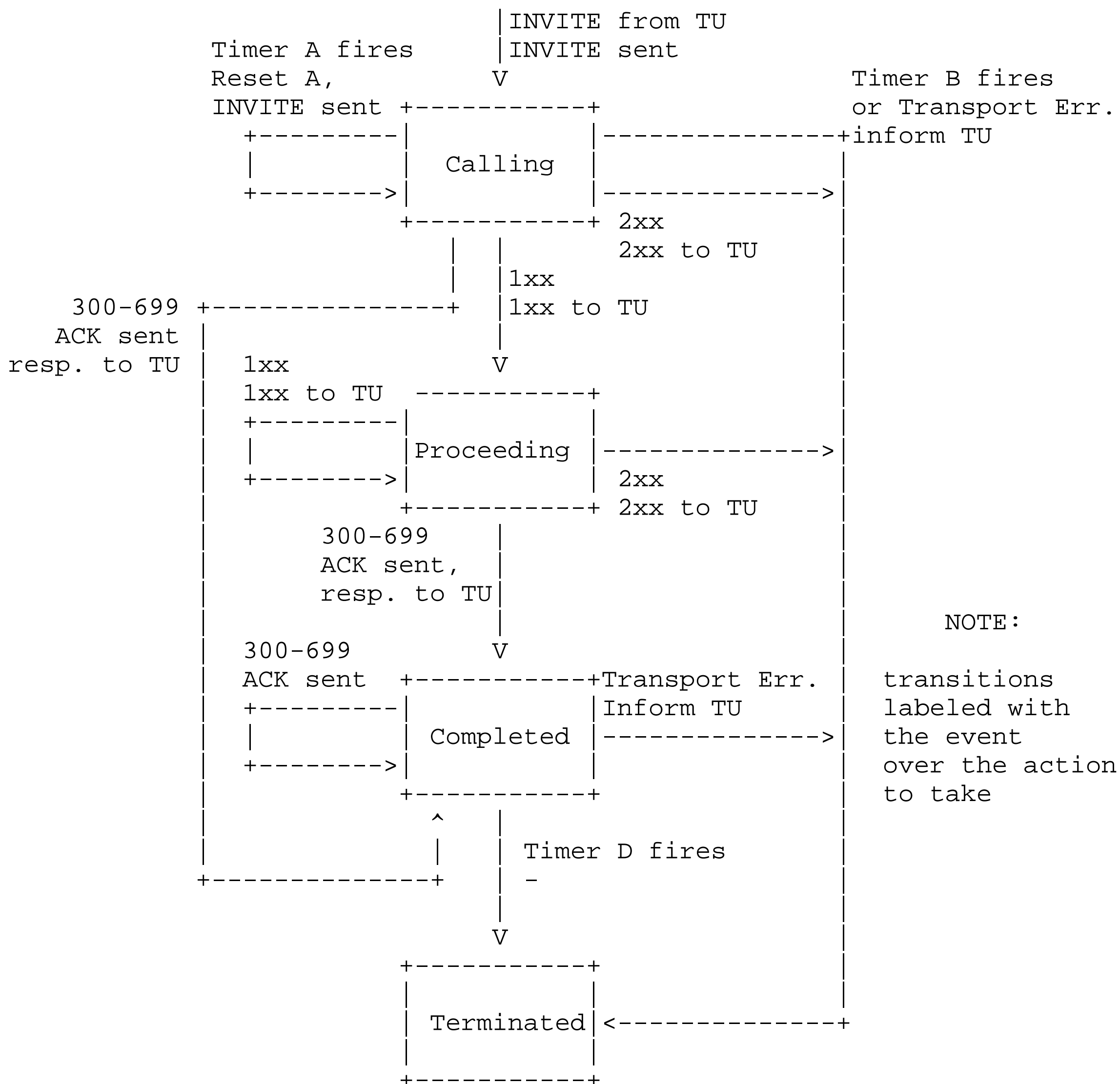


Figure 5: INVITE client transaction

If timer D fires while the client transaction is in the "Completed" state, the client transaction MUST move to the terminated state.

When in either the "Calling" or "Proceeding" states, reception of a 2xx response MUST cause the client transaction to enter the "Terminated" state, and the response MUST be passed up to the TU. The handling of this response depends on whether the TU is a proxy

core or a UAC core. A UAC core will handle generation of the ACK for this response, while a proxy core will always forward the 200 (OK) upstream. The differing treatment of 200 (OK) between proxy and UAC is the reason that handling of it does not take place in the transaction layer.

The client transaction MUST be destroyed the instant it enters the "Terminated" state. This is actually necessary to guarantee correct operation. The reason is that 2xx responses to an INVITE are treated differently; each one is forwarded by proxies, and the ACK handling in a UAC is different. Thus, each 2xx needs to be passed to a proxy core (so that it can be forwarded) and to a UAC core (so it can be acknowledged). No transaction layer processing takes place. Whenever a response is received by the transport, if the transport layer finds no matching client transaction (using the rules of Section 17.1.3), the response is passed directly to the core. Since the matching client transaction is destroyed by the first 2xx, subsequent 2xx will find no match and therefore be passed to the core.

17.1.1.3 Construction of the ACK Request

This section specifies the construction of ACK requests sent within the client transaction. A UAC core that generates an ACK for 2xx MUST instead follow the rules described in Section 13.

The ACK request constructed by the client transaction MUST contain values for the Call-ID, From, and Request-URI that are equal to the values of those header fields in the request passed to the transport by the client transaction (call this the "original request"). The To header field in the ACK MUST equal the To header field in the response being acknowledged, and therefore will usually differ from the To header field in the original request by the addition of the tag parameter. The ACK MUST contain a single Via header field, and this MUST be equal to the top Via header field of the original request. The CSeq header field in the ACK MUST contain the same value for the sequence number as was present in the original request, but the method parameter MUST be equal to "ACK".

If the INVITE request whose response is being acknowledged had Route header fields, those header fields MUST appear in the ACK. This is to ensure that the ACK can be routed properly through any downstream stateless proxies.

Although any request MAY contain a body, a body in an ACK is special since the request cannot be rejected if the body is not understood. Therefore, placement of bodies in ACK for non-2xx is NOT RECOMMENDED, but if done, the body types are restricted to any that appeared in the INVITE, assuming that the response to the INVITE was not 415. If it was, the body in the ACK MAY be any type listed in the Accept header field in the 415.

For example, consider the following request:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
```

The ACK request for a non-2xx final response to this request would look like this:

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 ACK
```

17.1.2 Non-INVITE Client Transaction

17.1.2.1 Overview of the non-INVITE Transaction

Non-INVITE transactions do not make use of ACK. They are simple request-response interactions. For unreliable transports, requests are retransmitted at an interval which starts at T1 and doubles until it hits T2. If a provisional response is received, retransmissions continue for unreliable transports, but at an interval of T2. The server transaction retransmits the last response it sent, which can be a provisional or final response, only when a retransmission of the request is received. This is why request retransmissions need to continue even after a provisional response; they are to ensure reliable delivery of the final response.

Unlike an INVITE transaction, a non-INVITE transaction has no special handling for the 2xx response. The result is that only a single 2xx response to a non-INVITE is ever delivered to a UAC.

17.1.2.2 Formal Description

The state machine for the non-INVITE client transaction is shown in Figure 6. It is very similar to the state machine for INVITE.

The "Trying" state is entered when the TU initiates a new client transaction with a request. When entering this state, the client transaction SHOULD set timer F to fire in 64*T1 seconds. The request MUST be passed to the transport layer for transmission. If an unreliable transport is in use, the client transaction MUST set timer E to fire in T1 seconds. If timer E fires while still in this state, the timer is reset, but this time with a value of MIN(2*T1, T2). When the timer fires again, it is reset to a MIN(4*T1, T2). This process continues so that retransmissions occur with an exponentially increasing interval that caps at T2. The default value of T2 is 4s, and it represents the amount of time a non-INVITE server transaction will take to respond to a request, if it does not respond immediately. For the default values of T1 and T2, this results in intervals of 500 ms, 1 s, 2 s, 4 s, 4 s, 4 s, etc.

If Timer F fires while the client transaction is still in the "Trying" state, the client transaction SHOULD inform the TU about the timeout, and then it SHOULD enter the "Terminated" state. If a provisional response is received while in the "Trying" state, the response MUST be passed to the TU, and then the client transaction SHOULD move to the "Proceeding" state. If a final response (status codes 200-699) is received while in the "Trying" state, the response MUST be passed to the TU, and the client transaction MUST transition to the "Completed" state.

If Timer E fires while in the "Proceeding" state, the request MUST be passed to the transport layer for retransmission, and Timer E MUST be reset with a value of T2 seconds. If timer F fires while in the "Proceeding" state, the TU MUST be informed of a timeout, and the client transaction MUST transition to the terminated state. If a final response (status codes 200-699) is received while in the "Proceeding" state, the response MUST be passed to the TU, and the client transaction MUST transition to the "Completed" state.

Once the client transaction enters the "Completed" state, it MUST set Timer K to fire in T4 seconds for unreliable transports, and zero seconds for reliable transports. The "Completed" state exists to buffer any additional response retransmissions that may be received (which is why the client transaction remains there only for

unreliable transports). T4 represents the amount of time the network will take to clear messages between client and server transactions. The default value of T4 is 5s. A response is a retransmission when it matches the same transaction, using the rules specified in Section 17.1.3. If Timer K fires while in this state, the client transaction MUST transition to the "Terminated" state.

Once the transaction is in the terminated state, it MUST be destroyed immediately.

17.1.3 Matching Responses to Client Transactions

When the transport layer in the client receives a response, it has to determine which client transaction will handle the response, so that the processing of Sections 17.1.1 and 17.1.2 can take place. The branch parameter in the top Via header field is used for this purpose. A response matches a client transaction under two conditions:

1. If the response has the same value of the branch parameter in the top Via header field as the branch parameter in the top Via header field of the request that created the transaction.

2. If the method parameter in the CSeq header field matches the method of the request that created the transaction. The method is needed since a CANCEL request constitutes a different transaction, but shares the same value of the branch parameter.

If a request is sent via multicast, it is possible that it will generate multiple responses from different servers. These responses will all have the same branch parameter in the topmost Via, but vary in the To tag. The first response received, based on the rules above, will be used, and others will be viewed as retransmissions. That is not an error; multicast SIP provides only a rudimentary "single-hop-discovery-like" service that is limited to processing a single response. See Section 18.1.1 for details.

17.1.4 Handling Transport Errors

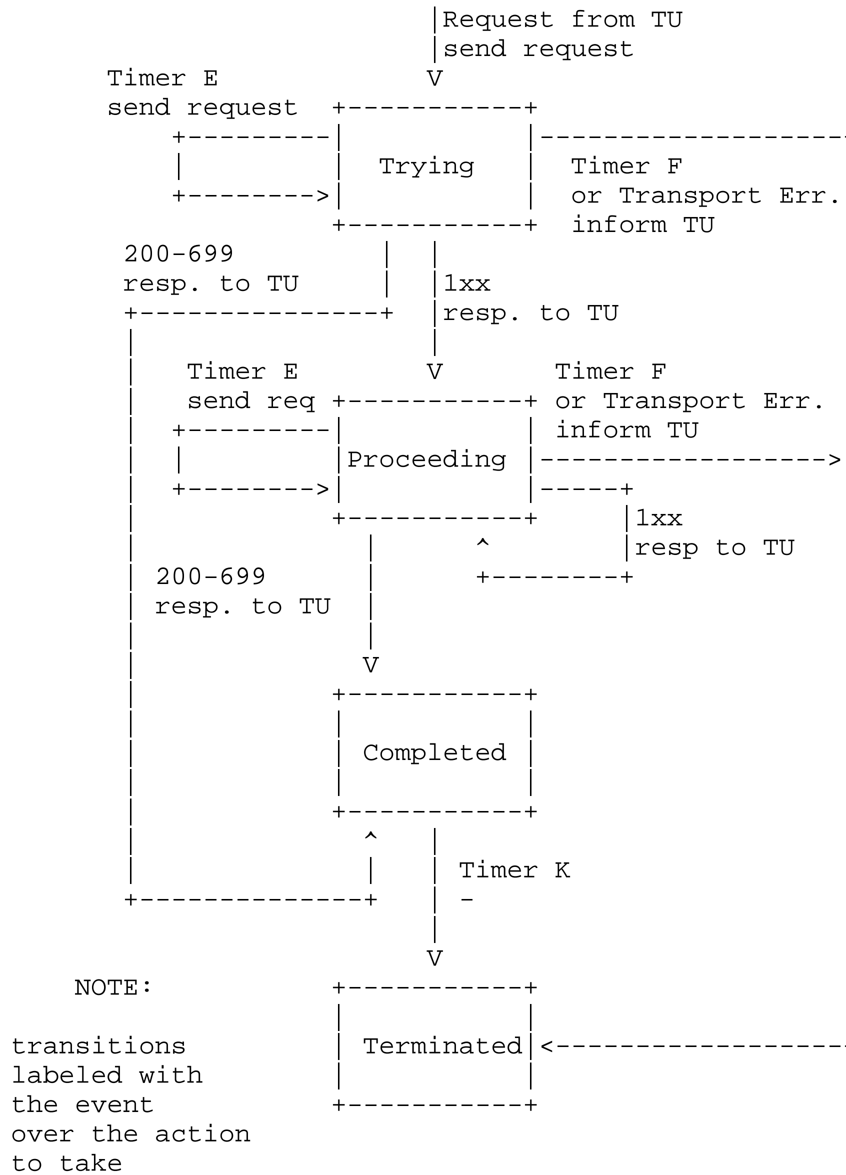


Figure 6: non-INVITE client transaction

When the client transaction sends a request to the transport layer to be sent, the following procedures are followed if the transport layer indicates a failure.

The client transaction SHOULD inform the TU that a transport failure has occurred, and the client transaction SHOULD transition directly to the "Terminated" state. The TU will handle the failover mechanisms described in [4].

17.2 Server Transaction

The server transaction is responsible for the delivery of requests to the TU and the reliable transmission of responses. It accomplishes this through a state machine. Server transactions are created by the core when a request is received, and transaction handling is desired for that request (this is not always the case).

As with the client transactions, the state machine depends on whether the received request is an INVITE request.

17.2.1 INVITE Server Transaction

The state diagram for the INVITE server transaction is shown in Figure 7.

When a server transaction is constructed for a request, it enters the "Proceeding" state. The server transaction MUST generate a 100 (Trying) response unless it knows that the TU will generate a provisional or final response within 200 ms, in which case it MAY generate a 100 (Trying) response. This provisional response is needed to quench request retransmissions rapidly in order to avoid network congestion. The 100 (Trying) response is constructed according to the procedures in Section 8.2.6, except that the insertion of tags in the To header field of the response (when none was present in the request) is downgraded from MAY to SHOULD NOT. The request MUST be passed to the TU.

The TU passes any number of provisional responses to the server transaction. So long as the server transaction is in the "Proceeding" state, each of these MUST be passed to the transport layer for transmission. They are not sent reliably by the transaction layer (they are not retransmitted by it) and do not cause a change in the state of the server transaction. If a request retransmission is received while in the "Proceeding" state, the most recent provisional response that was received from the TU MUST be passed to the transport layer for retransmission. A request is a retransmission if it matches the same server transaction based on the rules of Section 17.2.3.

If, while in the "Proceeding" state, the TU passes a 2xx response to the server transaction, the server transaction MUST pass this response to the transport layer for transmission. It is not

retransmitted by the server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction MUST then transition to the "Terminated" state.

While in the "Proceeding" state, if the TU passes a response with status code from 300 to 699 to the server transaction, the response MUST be passed to the transport layer for transmission, and the state machine MUST enter the "Completed" state. For unreliable transports, timer G is set to fire in T1 seconds, and is not set to fire for reliable transports.

This is a change from RFC 2543, where responses were always retransmitted, even over reliable transports.

When the "Completed" state is entered, timer H MUST be set to fire in 64*T1 seconds for all transports. Timer H determines when the server transaction abandons retransmitting the response. Its value is chosen to equal Timer B, the amount of time a client transaction will continue to retry sending a request. If timer G fires, the response is passed to the transport layer once more for retransmission, and timer G is set to fire in MIN(2*T1, T2) seconds. From then on, when timer G fires, the response is passed to the transport again for transmission, and timer G is reset with a value that doubles, unless that value exceeds T2, in which case it is reset with the value of T2. This is identical to the retransmit behavior for requests in the "Trying" state of the non-INVITE client transaction. Furthermore, while in the "Completed" state, if a request retransmission is received, the server SHOULD pass the response to the transport for retransmission.

If an ACK is received while the server transaction is in the "Completed" state, the server transaction MUST transition to the "Confirmed" state. As Timer G is ignored in this state, any retransmissions of the response will cease.

If timer H fires while in the "Completed" state, it implies that the ACK was never received. In this case, the server transaction MUST transition to the "Terminated" state, and MUST indicate to the TU that a transaction failure has occurred.

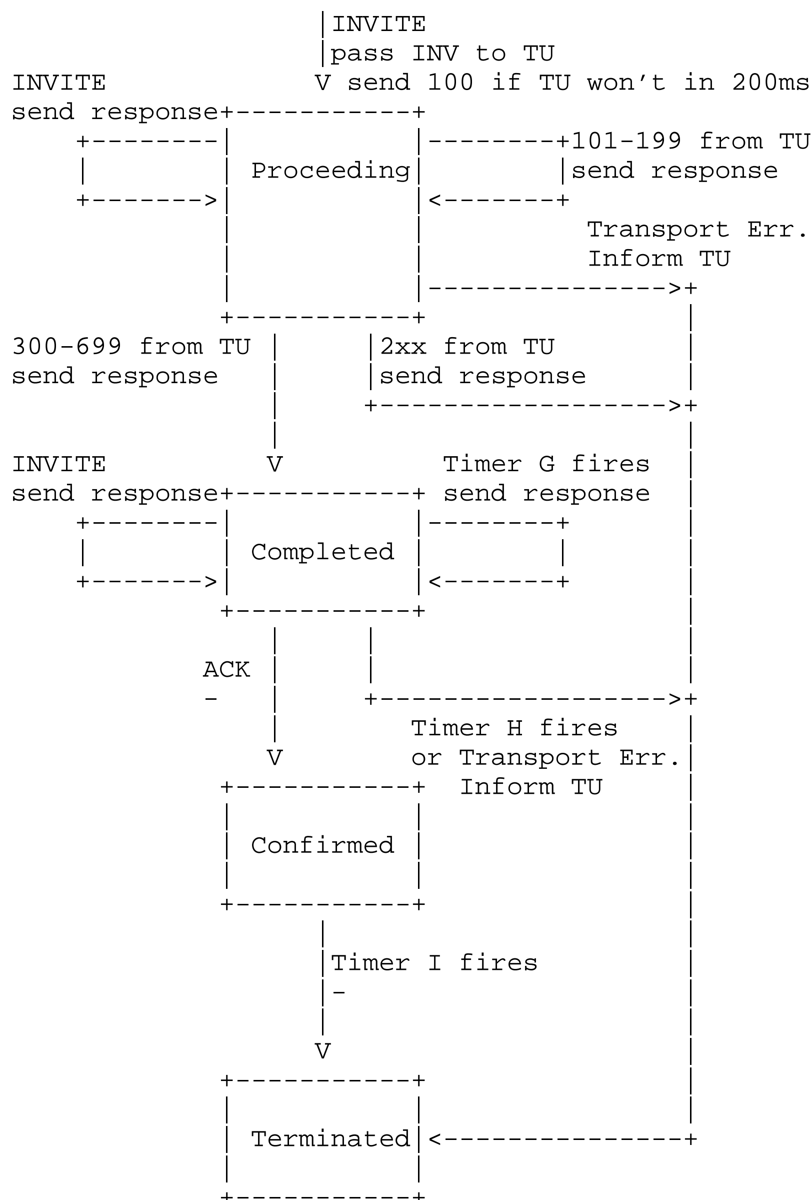


Figure 7: INVITE server transaction

The purpose of the "Confirmed" state is to absorb any additional ACK messages that arrive, triggered from retransmissions of the final response. When this state is entered, timer I is set to fire in T4 seconds for unreliable transports, and zero seconds for reliable transports. Once timer I fires, the server MUST transition to the "Terminated" state.

Once the transaction is in the "Terminated" state, it MUST be destroyed immediately. As with client transactions, this is needed to ensure reliability of the 2xx responses to INVITE.

17.2.2 Non-INVITE Server Transaction

The state machine for the non-INVITE server transaction is shown in Figure 8.

The state machine is initialized in the "Trying" state and is passed a request other than INVITE or ACK when initialized. This request is passed up to the TU. Once in the "Trying" state, any further request retransmissions are discarded. A request is a retransmission if it matches the same server transaction, using the rules specified in Section 17.2.3.

While in the "Trying" state, if the TU passes a provisional response to the server transaction, the server transaction MUST enter the "Proceeding" state. The response MUST be passed to the transport layer for transmission. Any further provisional responses that are received from the TU while in the "Proceeding" state MUST be passed to the transport layer for transmission. If a retransmission of the request is received while in the "Proceeding" state, the most recently sent provisional response MUST be passed to the transport layer for retransmission. If the TU passes a final response (status codes 200-699) to the server while in the "Proceeding" state, the transaction MUST enter the "Completed" state, and the response MUST be passed to the transport layer for transmission.

When the server transaction enters the "Completed" state, it MUST set Timer J to fire in 64*T1 seconds for unreliable transports, and zero seconds for reliable transports. While in the "Completed" state, the server transaction MUST pass the final response to the transport layer for retransmission whenever a retransmission of the request is received. Any other final responses passed by the TU to the server transaction MUST be discarded while in the "Completed" state. The server transaction remains in this state until Timer J fires, at which point it MUST transition to the "Terminated" state.

The server transaction MUST be destroyed the instant it enters the "Terminated" state.

17.2.3 Matching Requests to Server Transactions

When a request is received from the network by the server, it has to be matched to an existing transaction. This is accomplished in the following manner.

The branch parameter in the topmost Via header field of the request is examined. If it is present and begins with the magic cookie "z9hG4bK", the request was generated by a client transaction compliant to this specification. Therefore, the branch parameter will be unique across all transactions sent by that client. The request matches a transaction if:

1. the branch parameter in the request is equal to the one in the top Via header field of the request that created the transaction, and

2. the sent-by value in the top Via of the request is equal to the one in the request that created the transaction, and

3. the method of the request matches the one that created the transaction, except for ACK, where the method of the request that created the transaction is INVITE.

This matching rule applies to both INVITE and non-INVITE transactions alike.

> The sent-by value is used as part of the matching process because there could be accidental or malicious duplication of branch parameters from different clients.

If the branch parameter in the top Via header field is not present, or does not contain the magic cookie, the following procedures are used. These exist to handle backwards compatibility with RFC 2543 compliant implementations.

The INVITE request matches a transaction if the Request-URI, To tag, From tag, Call-ID, CSeq, and top Via header field match those of the INVITE request which created the transaction. In this case, the INVITE is a retransmission of the original one that created the transaction. The ACK request matches a transaction if the Request-URI, From tag, Call-ID, CSeq number (not the method), and top Via header field match those of the INVITE request which created the transaction, and the To tag of the ACK matches the To tag of the response sent by the server transaction. Matching is done based on the matching rules defined for each of those header fields. Inclusion of the tag in the To header field in the ACK matching process helps disambiguate ACK for 2xx from ACK for other responses

at a proxy, which may have forwarded both responses (This can occur in unusual conditions. Specifically, when a proxy forked a request, and then crashes, the responses may be delivered to another proxy, which might end up forwarding multiple responses upstream). An ACK request that matches an INVITE transaction matched by a previous ACK is considered a retransmission of that previous ACK.

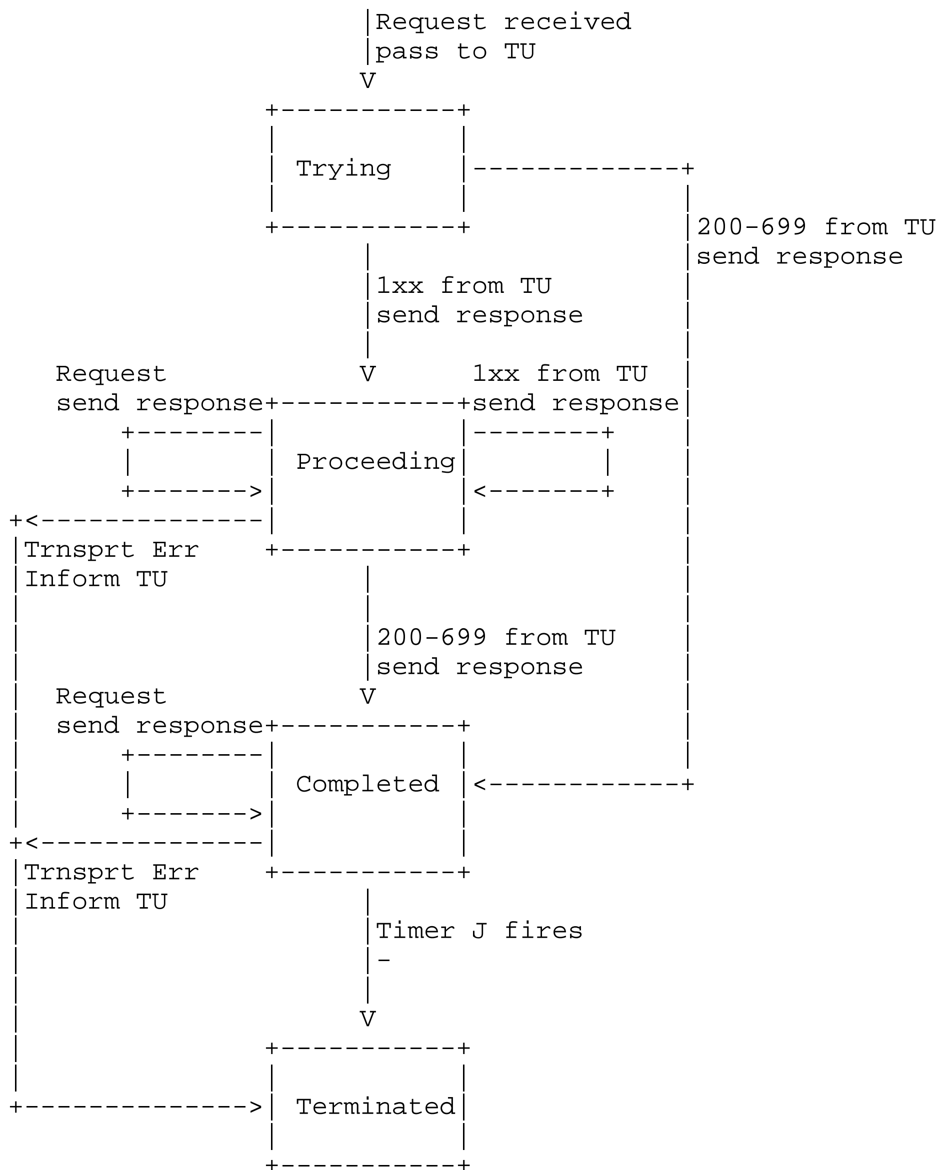


Figure 8: non-INVITE server transaction

For all other request methods, a request is matched to a transaction if the Request-URI, To tag, From tag, Call-ID, CSeq (including the method), and top Via header field match those of the request that created the transaction. Matching is done based on the matching

rules defined for each of those header fields. When a non-INVITE request matches an existing transaction, it is a retransmission of the request that created that transaction.

Because the matching rules include the Request-URI, the server cannot match a response to a transaction. When the TU passes a response to the server transaction, it must pass it to the specific server transaction for which the response is targeted.

17.2.4 Handling Transport Errors

When the server transaction sends a response to the transport layer to be sent, the following procedures are followed if the transport layer indicates a failure.

First, the procedures in [4] are followed, which attempt to deliver the response to a backup. If those should all fail, based on the definition of failure in [4], the server transaction SHOULD inform the TU that a failure has occurred, and SHOULD transition to the terminated state.

18 Transport

The transport layer is responsible for the actual transmission of requests and responses over network transports. This includes determination of the connection to use for a request or response in the case of connection-oriented transports.

The transport layer is responsible for managing persistent connections for transport protocols like TCP and SCTP, or TLS over those, including ones opened to the transport layer. This includes connections opened by the client or server transports, so that connections are shared between client and server transport functions. These connections are indexed by the tuple formed from the address, port, and transport protocol at the far end of the connection. When a connection is opened by the transport layer, this index is set to the destination IP, port and transport. When the connection is accepted by the transport layer, this index is set to the source IP address, port number, and transport. Note that, because the source port is often ephemeral, but it cannot be known whether it is ephemeral or selected through procedures in [4], connections accepted by the transport layer will frequently not be reused. The result is that two proxies in a "peering" relationship using a connection-oriented transport frequently will have two connections in use, one for transactions initiated in each direction.

It is RECOMMENDED that connections be kept open for some implementation-defined duration after the last message was sent or received over that connection. This duration SHOULD at least equal the longest amount of time the element would need in order to bring a transaction from instantiation to the terminated state. This is to make it likely that transactions are completed over the same connection on which they are initiated (for example, request, response, and in the case of INVITE, ACK for non-2xx responses). This usually means at least 64*T1 (see Section 17.1.1.1 for a definition of T1). However, it could be larger in an element that has a TU using a large value for timer C (bullet 11 of Section 16.6), for example.

All SIP elements MUST implement UDP and TCP. SIP elements MAY implement other protocols.

> Making TCP mandatory for the UA is a substantial change from RFC 2543. It has arisen out of the need to handle larger messages, which MUST use TCP, as discussed below. Thus, even if an element never sends large messages, it may receive one and needs to be able to handle them.

18.1 Clients

18.1.1 Sending Requests

The client side of the transport layer is responsible for sending the request and receiving responses. The user of the transport layer passes the client transport the request, an IP address, port, transport, and possibly TTL for multicast destinations.

If a request is within 200 bytes of the path MTU, or if it is larger than 1300 bytes and the path MTU is unknown, the request MUST be sent using an RFC 2914 [43] congestion controlled transport protocol, such as TCP. If this causes a change in the transport protocol from the one indicated in the top Via, the value in the top Via MUST be changed. This prevents fragmentation of messages over UDP and provides congestion control for larger messages. However, implementations MUST be able to handle messages up to the maximum datagram packet size. For UDP, this size is 65,535 bytes, including IP and UDP headers.

> The 200 byte "buffer" between the message size and the MTU accommodates the fact that the response in SIP can be larger than the request. This happens due to the addition of Record-Route header field values to the responses to INVITE, for example. With the extra buffer, the response can be about 170 bytes larger than the request, and still not be fragmented on IPv4 (about 30 bytes

is consumed by IP/UDP, assuming no IPSec). 1300 is chosen when path MTU is not known, based on the assumption of a 1500 byte Ethernet MTU.

If an element sends a request over TCP because of these message size constraints, and that request would have otherwise been sent over UDP, if the attempt to establish the connection generates either an ICMP Protocol Not Supported, or results in a TCP reset, the element SHOULD retry the request, using UDP. This is only to provide backwards compatibility with RFC 2543 compliant implementations that do not support TCP. It is anticipated that this behavior will be deprecated in a future revision of this specification.

A client that sends a request to a multicast address MUST add the "maddr" parameter to its Via header field value containing the destination multicast address, and for IPv4, SHOULD add the "ttl" parameter with a value of 1. Usage of IPv6 multicast is not defined in this specification, and will be a subject of future standardization when the need arises.

These rules result in a purposeful limitation of multicast in SIP. Its primary function is to provide a "single-hop-discovery-like" service, delivering a request to a group of homogeneous servers, where it is only required to process the response from any one of them. This functionality is most useful for registrations. In fact, based on the transaction processing rules in Section 17.1.3, the client transaction will accept the first response, and view any others as retransmissions because they all contain the same Via branch identifier.

Before a request is sent, the client transport MUST insert a value of the "sent-by" field into the Via header field. This field contains an IP address or host name, and port. The usage of an FQDN is RECOMMENDED. This field is used for sending responses under certain conditions, described below. If the port is absent, the default value depends on the transport. It is 5060 for UDP, TCP and SCTP, 5061 for TLS.

For reliable transports, the response is normally sent on the connection on which the request was received. Therefore, the client transport MUST be prepared to receive the response on the same connection used to send the request. Under error conditions, the server may attempt to open a new connection to send the response. To handle this case, the transport layer MUST also be prepared to receive an incoming connection on the source IP address from which the request was sent and port number in the "sent-by" field. It also

MUST be prepared to receive incoming connections on any address and port that would be selected by a server based on the procedures described in Section 5 of [4].

For unreliable unicast transports, the client transport MUST be prepared to receive responses on the source IP address from which the request is sent (as responses are sent back to the source address) and the port number in the "sent-by" field. Furthermore, as with reliable transports, in certain cases the response will be sent elsewhere. The client MUST be prepared to receive responses on any address and port that would be selected by a server based on the procedures described in Section 5 of [4].

For multicast, the client transport MUST be prepared to receive responses on the same multicast group and port to which the request is sent (that is, it needs to be a member of the multicast group it sent the request to.)

If a request is destined to an IP address, port, and transport to which an existing connection is open, it is RECOMMENDED that this connection be used to send the request, but another connection MAY be opened and used.

If a request is sent using multicast, it is sent to the group address, port, and TTL provided by the transport user. If a request is sent using unicast unreliable transports, it is sent to the IP address and port provided by the transport user.

18.1.2 Receiving Responses

When a response is received, the client transport examines the top Via header field value. If the value of the "sent-by" parameter in that header field value does not correspond to a value that the client transport is configured to insert into requests, the response MUST be silently discarded.

If there are any client transactions in existence, the client transport uses the matching procedures of Section 17.1.3 to attempt to match the response to an existing transaction. If there is a match, the response MUST be passed to that transaction. Otherwise, the response MUST be passed to the core (whether it be stateless proxy, stateful proxy, or UA) for further processing. Handling of these "stray" responses is dependent on the core (a proxy will forward them, while a UA will discard, for example).

18.2 Servers

18.2.1 Receiving Requests

A server SHOULD be prepared to receive requests on any IP address, port and transport combination that can be the result of a DNS lookup on a SIP or SIPS URI [4] that is handed out for the purposes of communicating with that server. In this context, "handing out" includes placing a URI in a Contact header field in a REGISTER request or a redirect response, or in a Record-Route header field in a request or response. A URI can also be "handed out" by placing it on a web page or business card. It is also RECOMMENDED that a server listen for requests on the default SIP ports (5060 for TCP and UDP, 5061 for TLS over TCP) on all public interfaces. The typical exception would be private networks, or when multiple server instances are running on the same host. For any port and interface that a server listens on for UDP, it MUST listen on that same port and interface for TCP. This is because a message may need to be sent using TCP, rather than UDP, if it is too large. As a result, the converse is not true. A server need not listen for UDP on a particular address and port just because it is listening on that same address and port for TCP. There may, of course, be other reasons why a server needs to listen for UDP on a particular address and port.

When the server transport receives a request over any transport, it MUST examine the value of the "sent-by" parameter in the top Via header field value. If the host portion of the "sent-by" parameter contains a domain name, or if it contains an IP address that differs from the packet source address, the server MUST add a "received" parameter to that Via header field value. This parameter MUST contain the source address from which the packet was received. This is to assist the server transport layer in sending the response, since it must be sent to the source IP address from which the request came.

Consider a request received by the server transport which looks like, in part:

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

The request is received with a source IP address of 192.0.2.4. Before passing the request up, the transport adds a "received" parameter, so that the request would look like, in part:

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4
```

Next, the server transport attempts to match the request to a server transaction. It does so using the matching rules described in Section 17.2.3. If a matching server transaction is found, the request is passed to that transaction for processing. If no match is found, the request is passed to the core, which may decide to construct a new server transaction for that request. Note that when a UAS core sends a 2xx response to INVITE, the server transaction is destroyed. This means that when the ACK arrives, there will be no matching server transaction, and based on this rule, the ACK is passed to the UAS core, where it is processed.

18.2.2 Sending Responses

The server transport uses the value of the top Via header field in order to determine where to send a response. It MUST follow the following process:

- o If the "sent-protocol" is a reliable transport protocol such as TCP or SCTP, or TLS over those, the response MUST be sent using the existing connection to the source of the original request that created the transaction, if that connection is still open. This requires the server transport to maintain an association between server transactions and transport connections. If that connection is no longer open, the server SHOULD open a connection to the IP address in the "received" parameter, if present, using the port in the "sent-by" value, or the default port for that transport, if no port is specified. If that connection attempt fails, the server SHOULD use the procedures in [4] for servers in order to determine the IP address and port to open the connection and send the response to.

- o Otherwise, if the Via header field value contains a "maddr" parameter, the response MUST be forwarded to the address listed there, using the port indicated in "sent-by", or port 5060 if none is present. If the address is a multicast address, the response SHOULD be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present.

- o Otherwise (for unreliable unicast transports), if the top Via has a "received" parameter, the response MUST be sent to the address in the "received" parameter, using the port indicated in the "sent-by" value, or using port 5060 if none is specified explicitly. If this fails, for example, elicits an ICMP "port unreachable" response, the procedures of Section 5 of [4] SHOULD be used to determine where to send the response.

- o Otherwise, if it is not receiver-tagged, the response MUST be sent to the address indicated by the "sent-by" value, using the procedures in Section 5 of [4].

18.3 Framing

In the case of message-oriented transports (such as UDP), if the message has a Content-Length header field, the message body is assumed to contain that many bytes. If there are additional bytes in the transport packet beyond the end of the body, they MUST be discarded. If the transport packet ends before the end of the message body, this is considered an error. If the message is a response, it MUST be discarded. If the message is a request, the element SHOULD generate a 400 (Bad Request) response. If the message has no Content-Length header field, the message body is assumed to end at the end of the transport packet.

In the case of stream-oriented transports such as TCP, the Content-Length header field indicates the size of the body. The Content-Length header field MUST be used with stream oriented transports.

18.4 Error Handling

Error handling is independent of whether the message was a request or response.

If the transport user asks for a message to be sent over an unreliable transport, and the result is an ICMP error, the behavior depends on the type of ICMP error. Host, network, port or protocol unreachable errors, or parameter problem errors SHOULD cause the transport layer to inform the transport user of a failure in sending. Source quench and TTL exceeded ICMP errors SHOULD be ignored.

If the transport user asks for a request to be sent over a reliable transport, and the result is a connection failure, the transport layer SHOULD inform the transport user of a failure in sending.

19 Common Message Components

There are certain components of SIP messages that appear in various places within SIP messages (and sometimes, outside of them) that merit separate discussion.

19.1 SIP and SIPS Uniform Resource Indicators

A SIP or SIPS URI identifies a communications resource. Like all URIs, SIP and SIPS URIs may be placed in web pages, email messages, or printed literature. They contain sufficient information to initiate and maintain a communication session with the resource.

Examples of communications resources include the following:

- a user of an online service
- an appearance on a multi-line phone
- a mailbox on a messaging system
- a PSTN number at a gateway service
- a group (such as "sales" or "helpdesk") in an organization

A SIPS URI specifies that the resource be contacted securely. This means, in particular, that TLS is to be used between the UAC and the domain that owns the URI. From there, secure communications are used to reach the user, where the specific security mechanism depends on the policy of the domain. Any resource described by a SIP URI can be "upgraded" to a SIPS URI by just changing the scheme, if it is desired to communicate with that resource securely.

19.1.1 SIP and SIPS URI Components

The "sip:" and "sips:" schemes follow the guidelines in RFC 2396 [5]. They use a form similar to the mailto URL, allowing the specification of SIP request-header fields and the SIP message-body. This makes it possible to specify the subject, media type, or urgency of sessions initiated by using a URI on a web page or in an email message. The formal syntax for a SIP or SIPS URI is presented in Section 25. Its general form, in the case of a SIP URI, is:

```
sip:user:password@host:port;uri-parameters?headers
```

The format for a SIPS URI is the same, except that the scheme is "sips" instead of sip. These tokens, and some of the tokens in their expansions, have the following meanings:

user: The identifier of a particular resource at the host being addressed. The term "host" in this context frequently refers to a domain. The "userinfo" of a URI consists of this user field, the password field, and the @ sign following them. The userinfo part of a URI is optional and MAY be absent when the

destination host does not have a notion of users or when the host itself is the resource being identified. If the @ sign is present in a SIP or SIPS URI, the user field MUST NOT be empty.

If the host being addressed can process telephone numbers, for instance, an Internet telephony gateway, a telephone-subscriber field defined in RFC 2806 [9] MAY be used to populate the user field. There are special escaping rules for encoding telephone-subscriber fields in SIP and SIPS URIs described in Section 19.1.2.

password: A password associated with the user. While the SIP and SIPS URI syntax allows this field to be present, its use is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URIs) has proven to be a security risk in almost every case where it has been used. For instance, transporting a PIN number in this field exposes the PIN.

Note that the password field is just an extension of the user portion. Implementations not wishing to give special significance to the password portion of the field MAY simply treat "user:password" as a single string.

host: The host providing the SIP resource. The host part contains either a fully-qualified domain name or numeric IPv4 or IPv6 address. Using the fully-qualified domain name form is RECOMMENDED whenever possible.

port: The port number where the request is to be sent.

URI parameters: Parameters affecting a request constructed from the URI.

URI parameters are added after the hostport component and are separated by semi-colons.

URI parameters take the form:

```
parameter-name "=" parameter-value
```

Even though an arbitrary number of URI parameters may be included in a URI, any given parameter-name MUST NOT appear more than once.

This extensible mechanism includes the transport, maddr, ttl, user, method and lr parameters.

The transport parameter determines the transport mechanism to be used for sending SIP messages, as specified in [4]. SIP can use any network transport protocol. Parameter names are defined for UDP (RFC 768 [14]), TCP (RFC 761 [15]), and SCTP (RFC 2960 [16]). For a SIPS URI, the transport parameter MUST indicate a reliable transport.

The maddr parameter indicates the server address to be contacted for this user, overriding any address derived from the host field. When an maddr parameter is present, the port and transport components of the URI apply to the address indicated in the maddr parameter value. [4] describes the proper interpretation of the transport, maddr, and hostport in order to obtain the destination address, port, and transport for sending a request.

The maddr field has been used as a simple form of loose source routing. It allows a URI to specify a proxy that must be traversed en-route to the destination. Continuing to use the maddr parameter this way is strongly discouraged (the mechanisms that enable it are deprecated). Implementations should instead use the Route mechanism described in this document, establishing a pre-existing route set if necessary (see Section 8.1.1.1). This provides a full URI to describe the node to be traversed.

The ttl parameter determines the time-to-live value of the UDP multicast packet and MUST only be used if maddr is a multicast address and the transport protocol is UDP. For example, to specify a call to alice@atlanta.com using multicast to 239.255.255.1 with a ttl of 15, the following URI would be used:

```
sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15
```

The set of valid telephone-subscriber strings is a subset of valid user strings. The user URI parameter exists to distinguish telephone numbers from user names that happen to look like telephone numbers. If the user string contains a telephone number formatted as a telephone-subscriber, the user parameter value "phone" SHOULD be present. Even without this parameter, recipients of SIP and SIPS URIs MAY interpret the pre-@ part as a telephone number if local restrictions on the name space for user name allow it.

The method of the SIP request constructed from the URI can be specified with the method parameter.

The lr parameter, when present, indicates that the element responsible for this resource implements the routing mechanisms specified in this document. This parameter will be used in the URIs proxies place into Record-Route header field values, and may appear in the URIs in a pre-existing route set.

This parameter is used to achieve backwards compatibility with systems implementing the strict-routing mechanisms of RFC 2543 and the rfc2543bis drafts up to bis-05. An element preparing to send a request based on a URI not containing this parameter can assume the receiving element implements strict-routing and reformat the message to preserve the information in the Request-URI.

Since the uri-parameter mechanism is extensible, SIP elements MUST silently ignore any uri-parameters that they do not understand.

Headers: Header fields to be included in a request constructed from the URI.

Headers fields in the SIP request can be specified with the "?" mechanism within a URI. The header names and values are encoded in ampersand separated hname = hvalue pairs. The special hname "body" indicates that the associated hvalue is the message-body of the SIP request.

Table 1 summarizes the use of SIP and SIPS URI components based on the context in which the URI appears. The external column describes URIs appearing anywhere outside of a SIP message, for instance on a web page or business card. Entries marked "m" are mandatory, those marked "o" are optional, and those marked "-" are not allowed. Elements processing URIs SHOULD ignore any disallowed components if they are present. The second column indicates the default value of an optional element if it is not present. "--" indicates that the element is either not optional, or has no default value.

URIs in Contact header fields have different restrictions depending on the context in which the header field appears. One set applies to messages that establish and maintain dialogs (INVITE and its 200 (OK) response). The other applies to registration and redirection messages (REGISTER, its 200 (OK) response, and 3xx class responses to any method).

19.1.2 Character Escaping Requirements

	default	Req.-URI	To	From	reg./redir. Contact	dialog Contact/ R-R/Route	external
user	--	o	o	o	o	o	o
password	--	o	o	o	o	o	o
host	--	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	o
maddr-param	--	o	-	-	o	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	(2)	o	-	-	o	o	o
lr-param	--	o	-	-	-	o	o
other-param	--	o	o	o	o	o	o
headers	--	-	-	-	o	-	o

(1): The default port value is transport and scheme dependent. The default is 5060 for sip: using UDP, TCP, or SCTP. The default is 5061 for sip: using TLS over TCP and sips: over TCP.

(2): The default transport is scheme dependent. For sip:, it is UDP. For sips:, it is TCP.

Table 1: Use and default values of URI components for SIP header field values, Request-URI and references

SIP follows the requirements and guidelines of RFC 2396 [5] when defining the set of characters that must be escaped in a SIP URI, and uses its ""%" HEX HEX" mechanism for escaping. From RFC 2396 [5]:

> The set of characters actually reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding [5]. Excluded US-ASCII characters (RFC 2396 [5]), such as space and control characters and characters used as URI delimiters, also MUST be escaped. URIs MUST NOT contain unescaped space and control characters.

For each component, the set of valid BNF expansions defines exactly which characters may appear unescaped. All other characters MUST be escaped.

For example, "@" is not in the set of characters in the user component, so the user "j@s0n" must have at least the @ sign encoded, as in "j%40s0n".

Expanding the hname and hvalue tokens in Section 25 show that all URI reserved characters in header field names and values MUST be escaped.

The telephone-subscriber subset of the user component has special escaping considerations. The set of characters not reserved in the RFC 2806 [9] description of telephone-subscriber contains a number of characters in various syntax elements that need to be escaped when used in SIP URIs. Any characters occurring in a telephone-subscriber that do not appear in an expansion of the BNF for the user rule MUST be escaped.

Note that character escaping is not allowed in the host component of a SIP or SIPS URI (the % character is not valid in its expansion). This is likely to change in the future as requirements for Internationalized Domain Names are finalized. Current implementations MUST NOT attempt to improve robustness by treating received escaped characters in the host component as literally equivalent to their unescaped counterpart. The behavior required to meet the requirements of IDN may be significantly different.

19.1.3 Example SIP and SIPS URIs

```
sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sips:alice@atlanta.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sips:1212@gateway.com
sip:alice@192.0.2.4
sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
sip:alice;day=tuesday@atlanta.com
```

The last sample URI above has a user field value of "alice;day=tuesday". The escaping rules defined above allow a semicolon to appear unescaped in this field. For the purposes of this protocol, the field is opaque. The structure of that value is only useful to the SIP element responsible for the resource.

19.1.4 URI Comparison

Some operations in this specification require determining whether two SIP or SIPS URIs are equivalent. In this specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). SIP and SIPS URIs are compared for equality according to the following rules:

- o A SIP and SIPS URI are never equivalent.

- o Comparison of the userinfo of SIP and SIPS URIs is case-sensitive. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is case-insensitive unless explicitly defined otherwise.

- o The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.

- o Characters other than those in the "reserved" set (see RFC 2396 [5]) are equivalent to their ""%" HEX HEX" encoding.

- o An IP address that is the result of a DNS lookup of a host name does not match that host name.

- o For two URIs to be equal, the user, password, host, and port components must match.

  A URI omitting the user component will not match a URI that includes one. A URI omitting the password component will not match a URI that includes one.

  A URI omitting any component with a default value will not match a URI explicitly containing that component with its default value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060. The same is true for the transport-parameter, ttl-parameter, user-parameter, and method components.

  > Defining sip:user@host to not be equivalent to sip:user@host:5060 is a change from RFC 2543. When deriving addresses from URIs, equivalent addresses are expected from equivalent URIs. The URI sip:user@host:5060 will always resolve to port 5060. The URI sip:user@host may resolve to other ports through the DNS SRV mechanisms detailed in [4].

- o URI uri-parameter components are compared as follows:

  - Any uri-parameter appearing in both URIs must match.

  - A user, ttl, or method uri-parameter appearing in only one URI never matches, even if it contains the default value.

  - A URI that includes an maddr parameter will not match a URI that contains no maddr parameter.

  - All other uri-parameters appearing in only one URI are ignored when comparing the URIs.

- o URI header components are never ignored. Any present header component MUST be present in both URIs and match for the URIs to match. The matching rules are defined for each header field in Section 20.

The URIs within each of the following sets are equivalent:

```
sip:%61lice@atlanta.com;transport=TCP
sip:alice@AtLanTa.CoM;Transport=tcp

sip:carol@chicago.com
sip:carol@chicago.com;newparam=5
sip:carol@chicago.com;security=on

sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com
sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@atlanta.com?priority=urgent&subject=project%20x
```

The URIs within each of the following sets are not equivalent:

```
SIP:ALICE@AtLanTa.CoM;Transport=udp             (different usernames)
sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com                   (can resolve to different ports)
sip:bob@biloxi.com:5060

sip:bob@biloxi.com              (can resolve to different transports)
sip:bob@biloxi.com;transport=udp

sip:bob@biloxi.com     (can resolve to different port and transports)
sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com                    (different header component)
sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com   (even though that's what
sip:bob@192.0.2.4                 phone21.boxesbybob.com resolves to)
```

Note that equality is not transitive:

- o sip:carol@chicago.com and sip:carol@chicago.com;security=on are equivalent

- o sip:carol@chicago.com and sip:carol@chicago.com;security=off are equivalent

- o sip:carol@chicago.com;security=on and sip:carol@chicago.com;security=off are not equivalent

19.1.5 Forming Requests from a URI

An implementation needs to take care when forming requests directly from a URI. URIs from business cards, web pages, and even from sources inside the protocol such as registered contacts may contain inappropriate header fields or body parts.

An implementation MUST include any provided transport, maddr, ttl, or user parameter in the Request-URI of the formed request. If the URI contains a method parameter, its value MUST be used as the method of the request. The method parameter MUST NOT be placed in the Request-URI. Unknown URI parameters MUST be placed in the message's Request-URI.

An implementation SHOULD treat the presence of any headers or body parts in the URI as a desire to include them in the message, and choose to honor the request on a per-component basis.

An implementation SHOULD NOT honor these obviously dangerous header fields: From, Call-ID, CSeq, Via, and Record-Route.

An implementation SHOULD NOT honor any requested Route header field values in order to not be used as an unwitting agent in malicious attacks.

An implementation SHOULD NOT honor requests to include header fields that may cause it to falsely advertise its location or capabilities. These include: Accept, Accept-Encoding, Accept-Language, Allow, Contact (in its dialog usage), Organization, Supported, and User-Agent.

An implementation SHOULD verify the accuracy of any requested descriptive header fields, including: Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Type, Date, Mime-Version, and Timestamp.

If the request formed from constructing a message from a given URI is not a valid SIP request, the URI is invalid. An implementation MUST NOT proceed with transmitting the request. It should instead pursue the course of action due an invalid URI in the context it occurs.

> The constructed request can be invalid in many ways. These include, but are not limited to, syntax error in header fields, invalid combinations of URI parameters, or an incorrect description of the message body.

Sending a request formed from a given URI may require capabilities unavailable to the implementation. The URI might indicate use of an unimplemented transport or extension, for example. An implementation SHOULD refuse to send these requests rather than modifying them to match their capabilities. An implementation MUST NOT send a request requiring an extension that it does not support.

For example, such a request can be formed through the presence of a Require header parameter or a method URI parameter with an unknown or explicitly unsupported value.

19.1.6 Relating SIP URIs and tel URLs

When a tel URL (RFC 2806 [9]) is converted to a SIP or SIPS URI, the entire telephone-subscriber portion of the tel URL, including any parameters, is placed into the userinfo part of the SIP or SIPS URI.

Thus, tel:+358-555-1234567;postd=pp22 becomes

```
sip:+358-555-1234567;postd=pp22@foo.com;user=phone
```

or

```
sips:+358-555-1234567;postd=pp22@foo.com;user=phone
```

not

```
sip:+358-555-1234567@foo.com;postd=pp22;user=phone
```

or

```
sips:+358-555-1234567@foo.com;postd=pp22;user=phone
```

In general, equivalent "tel" URLs converted to SIP or SIPS URIs in this fashion may not produce equivalent SIP or SIPS URIs. The userinfo of SIP and SIPS URIs are compared as a case-sensitive string. Variance in case-insensitive portions of tel URLs and reordering of tel URL parameters does not affect tel URL equivalence, but does affect the equivalence of SIP URIs formed from them.

For example,

```
tel:+358-555-1234567;postd=pp22
tel:+358-555-1234567;POSTD=PP22
```

are equivalent, while

```
sip:+358-555-1234567;postd=pp22@foo.com;user=phone
sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone
```

are not.

Likewise,

```
tel:+358-555-1234567;postd=pp22;isub=1411
tel:+358-555-1234567;isub=1411;postd=pp22
```

are equivalent, while

```
sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone
sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone
```

are not.

To mitigate this problem, elements constructing telephone-subscriber fields to place in the userinfo part of a SIP or SIPS URI SHOULD fold any case-insensitive portion of telephone-subscriber to lower case, and order the telephone-subscriber parameters lexically by parameter name, excepting isdn-subaddress and post-dial, which occur first and in that order. (All components of a tel URL except for future-extension parameters are defined to be compared case-insensitive.)

Following this suggestion, both

```
tel:+358-555-1234567;postd=pp22
tel:+358-555-1234567;POSTD=PP22
```

become

```
sip:+358-555-1234567;postd=pp22@foo.com;user=phone
```

and both

```
tel:+358-555-1234567;tsp=a.b;phone-context=5
tel:+358-555-1234567;phone-context=5;tsp=a.b
```

become

```
sip:+358-555-1234567;phone-context=5;tsp=a.b@foo.com;user=phone
```

19.2 Option Tags

Option tags are unique identifiers used to designate new options (extensions) in SIP. These tags are used in Require (Section 20.32), Proxy-Require (Section 20.29), Supported (Section 20.37) and Unsupported (Section 20.40) header fields. Note that these options appear as parameters in those header fields in an option-tag = token form (see Section 25 for the definition of token).

Option tags are defined in standards track RFCs. This is a change from past practice, and is instituted to ensure continuing multi-vendor interoperability (see discussion in Section 20.32 and Section 20.37). An IANA registry of option tags is used to ensure easy reference.

19.3 Tags

The "tag" parameter is used in the To and From header fields of SIP messages. It serves as a general mechanism to identify a dialog, which is the combination of the Call-ID along with two tags, one from each participant in the dialog. When a UA sends a request outside of a dialog, it contains a From tag only, providing "half" of the dialog ID. The dialog is completed from the response(s), each of which contributes the second half in the To header field. The forking of SIP requests means that multiple dialogs can be established from a single request. This also explains the need for the two-sided dialog identifier; without a contribution from the recipients, the originator could not disambiguate the multiple dialogs established from a single request.

When a tag is generated by a UA for insertion into a request or response, it MUST be globally unique and cryptographically random with at least 32 bits of randomness. A property of this selection requirement is that a UA will place a different tag into the From header of an INVITE than it would place into the To header of the response to the same INVITE. This is needed in order for a UA to invite itself to a session, a common case for "hairpinning" of calls in PSTN gateways. Similarly, two INVITEs for different calls will have different From tags, and two responses for different calls will have different To tags.

Besides the requirement for global uniqueness, the algorithm for generating a tag is implementation-specific. Tags are helpful in fault tolerant systems, where a dialog is to be recovered on an alternate server after a failure. A UAS can select the tag in such a way that a backup can recognize a request as part of a dialog on the failed server, and therefore determine that it should attempt to recover the dialog and any other state associated with it.

20 Header Fields

The general syntax for header fields is covered in Section 7.3. This section lists the full set of header fields along with notes on syntax, meaning, and usage. Throughout this section, we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification RFC 2616 [8]. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Tables 2 and 3.

The "where" column describes the request and response types in which the header field can be used. Values in this column are:

R: header field may only appear in requests;

r: header field may only appear in responses;

2xx, 4xx, etc.: A numerical value or range indicates response codes with which the header field can be used;

c: header field is copied from the request to the response.

An empty entry in the "where" column indicates that the header field may be present in all requests and responses.

The "proxy" column describes the operations a proxy may perform on a header field:

a: A proxy can add or concatenate the header field if not present.

m: A proxy can modify an existing header field value.

d: A proxy can delete a header field value.

r: A proxy must be able to read the header field, and thus this header field cannot be encrypted.

The next six columns relate to the presence of a header field in a method:

c: Conditional; requirements on the header field depend on the context of the message.

m: The header field is mandatory.

m*: The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without that header field.

o: The header field is optional.

t: The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without that header field.

If a stream-based protocol (such as TCP) is used as a transport, then the header field MUST be sent.

*: The header field is required if the message body is not empty. See Sections 20.14, 20.15 and 7.4 for details.

-: The header field is not applicable.

"Optional" means that an element MAY include the header field in a request or response, and a UA MAY ignore the header field if present in the request or response (The exception to this rule is the Require header field discussed in 20.32). A "mandatory" header field MUST be present in a request, and MUST be understood by the UAS receiving the request. A mandatory response header field MUST be present in the response, and the header field MUST be understood by the UAC processing the response. "Not applicable" means that the header field MUST NOT be present in a request. If one is placed in a request by mistake, it MUST be ignored by the UAS receiving the request. Similarly, a header field labeled "not applicable" for a response means that the UAS MUST NOT place the header field in the response, and the UAC MUST ignore the header field in the response.

A UA SHOULD ignore extension header parameters that are not understood.

A compact form of some common header field names is also defined for use when overall message size is an issue.

The Contact, From, and To header fields contain a URI. If the URI contains a comma, question mark or semicolon, the URI MUST be enclosed in angle brackets (< and >). Any URI parameters are contained within these brackets. If the URI is not enclosed in angle brackets, any semicolon-delimited parameters are header-parameters, not URI parameters.

20.1 Accept

The Accept header field follows the syntax defined in [H14.1]. The semantics are also identical, with the exception that if no Accept header field is present, the server SHOULD assume a default value of application/sdp.

An empty Accept header field means that no formats are acceptable.

Example:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	-	o	m*	o
Accept	2xx		-	-	-	o	m*	o
Accept	415		-	c	-	c	c	c
Accept-Encoding	R		-	o	-	o	o	o
Accept-Encoding	2xx		-	-	-	o	m*	o
Accept-Encoding	415		-	c	-	c	c	c
Accept-Language	R		-	o	-	o	o	o
Accept-Language	2xx		-	-	-	o	m*	o
Accept-Language	415		-	c	-	c	c	c
Alert-Info	R	ar	-	-	-	o	-	-
Alert-Info	180	ar	-	-	-	o	-	-
Allow	R		-	o	-	o	o	o
Allow	2xx		-	o	-	m*	m*	o
Allow	r		-	o	-	o	o	o
Allow	405		-	m	-	m	m	m
Authentication-Info	2xx		-	o	-	o	o	o
Authorization	R		o	o	o	o	o	o
Call-ID	c	r	m	m	m	m	m	m
Call-Info		ar	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	-	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx	d	-	o	-	o	o	o
Contact	485		-	o	-	o	o	o
Content-Disposition			o	o	-	o	o	o
Content-Encoding			o	o	-	o	o	o
Content-Language			o	o	-	o	o	o
Content-Length		ar	t	t	t	t	t	t
Content-Type			*	*	-	*	*	*
CSeq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699	a	-	o	o	o	o	o
Expires			-	-	-	o	-	o
From	c	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	amr	m	m	m	m	m	m
Min-Expires	423		-	-	-	-	-	m
MIME-Version			o	o	-	o	o	o
Organization		ar	-	-	-	o	o	o

Table 2: Summary of header fields, A--O

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-
Proxy-Authenticate	407	ar	-	m	-	m	m	m
Proxy-Authenticate	401	ar	-	o	o	o	o	o
Proxy-Authorization	R	dr	o	o	-	o	o	o
Proxy-Require	R	ar	-	o	-	o	o	o
Record-Route	R	ar	o	o	o	o	o	-
Record-Route	2xx,18x	mr	-	o	o	o	o	-
Reply-To			-	-	-	o	-	-
Require		ar	-	c	-	c	c	c
Retry-After	404,413,480,486		-	o	o	o	o	o
	500,503		-	o	o	o	o	o
	600,603		-	o	o	o	o	o
Route	R	adr	c	c	c	c	c	c
Server	r		-	o	o	o	o	o
Subject	R		-	-	-	o	-	-
Supported	R		-	o	o	m*	o	o
Supported	2xx		-	o	o	m*	m*	o
Timestamp			o	o	o	o	o	o
To	c(1)	r	m	m	m	m	m	m
Unsupported	420		-	m	-	m	m	m
User-Agent			o	o	o	o	o	o
Via	R	amr	m	m	m	m	m	m
Via	rc	dr	m	m	m	m	m	m
Warning	r		-	o	o	o	o	o
WWW-Authenticate	401	ar	-	m	-	m	m	m
WWW-Authenticate	407	ar	-	o	-	o	o	o

Table 3: Summary of header fields, P--Z; (1): copied with possible addition of tag

```
Accept: application/sdp;level=1, application/x-private, text/html
```

20.2 Accept-Encoding

The Accept-Encoding header field is similar to Accept, but restricts the content-codings [H3.5] that are acceptable in the response. See [H14.3]. The semantics in SIP are identical to those defined in [H14.3].

An empty Accept-Encoding header field is permissible. It is equivalent to Accept-Encoding: identity, that is, only the identity encoding, meaning no encoding, is permissible.

If no Accept-Encoding header field is present, the server SHOULD assume a default value of identity.

This differs slightly from the HTTP definition, which indicates that when not present, any encoding can be used, but the identity encoding is preferred.

Example:

```
Accept-Encoding: gzip
```

20.3 Accept-Language

The Accept-Language header field is used in requests to indicate the preferred languages for reason phrases, session descriptions, or status responses carried as message bodies in the response. If no Accept-Language header field is present, the server SHOULD assume all languages are acceptable to the client.

The Accept-Language header field follows the syntax defined in [H14.4]. The rules for ordering the languages based on the "q" parameter apply to SIP as well.

Example:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

20.4 Alert-Info

When present in an INVITE request, the Alert-Info header field specifies an alternative ring tone to the UAS. When present in a 180 (Ringing) response, the Alert-Info header field specifies an alternative ringback tone to the UAC. A typical usage is for a proxy to insert this header field to provide a distinctive ring feature.

The Alert-Info header field can introduce security risks. These risks and the ways to handle them are discussed in Section 20.9, which discusses the Call-Info header field since the risks are identical.

In addition, a user SHOULD be able to disable this feature selectively.

> This helps prevent disruptions that could result from the use of this header field by untrusted elements.

Example:

```
Alert-Info: <http://www.example.com/sounds/moo.wav>
```

20.5 Allow

The Allow header field lists the set of methods supported by the UA generating the message.

All methods, including ACK and CANCEL, understood by the UA MUST be included in the list of methods in the Allow header field, when present. The absence of an Allow header field MUST NOT be interpreted to mean that the UA sending the message supports no methods. Rather, it implies that the UA is not providing any information on what methods it supports.

Supplying an Allow header field in responses to methods other than OPTIONS reduces the number of messages needed.

Example:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

20.6 Authentication-Info

The Authentication-Info header field provides for mutual authentication with HTTP Digest. A UAS MAY include this header field in a 2xx response to a request that was successfully authenticated using digest based on the Authorization header field.

Syntax and semantics follow those specified in RFC 2617 [17].

Example:

```
Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"
```

20.7 Authorization

The Authorization header field contains authentication credentials of a UA. Section 22.2 overviews the use of the Authorization header field, and Section 22.4 describes the syntax and semantics when used with HTTP authentication.

This header field, along with Proxy-Authorization, breaks the general rules about multiple header field values. Although not a comma-separated list, this header field name may be present multiple times, and MUST NOT be combined into a single header line using the usual rules described in Section 7.3.

In the example below, there are no quotes around the Digest parameter:

```
Authorization: Digest username="Alice", realm="atlanta.com",
 nonce="84a4cc6f3082121f32b42a2187831a9e",
 response="7587245234b3434cc3412213e5f113a5432"
```

20.8 Call-ID

The Call-ID header field uniquely identifies a particular invitation or all registrations of a particular client. A single multimedia conference can give rise to several calls with different Call-IDs, for example, if a user invites a single individual several times to the same (long-running) conference. Call-IDs are case-sensitive and are simply compared byte-by-byte.

The compact form of the Call-ID header field is i.

Examples:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com
i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4
```

20.9 Call-Info

The Call-Info header field provides additional information about the caller or callee, depending on whether it is found in a request or response. The purpose of the URI is described by the "purpose" parameter. The "icon" parameter designates an image suitable as an iconic representation of the caller or callee. The "info" parameter describes the caller or callee in general, for example, through a web page. The "card" parameter provides a business card, for example, in vCard [36] or LDIF [37] formats. Additional tokens can be registered using IANA and the procedures in Section 27.

Use of the Call-Info header field can pose a security risk. If a callee fetches the URIs provided by a malicious caller, the callee may be at risk for displaying inappropriate or offensive content, dangerous or illegal content, and so on. Therefore, it is RECOMMENDED that a UA only render the information in the Call-Info header field if it can verify the authenticity of the element that originated the header field and trusts that element. This need not be the peer UA; a proxy can insert this header field into requests.

Example:

```
Call-Info: <http://wwww.example.com/alice/photo.jpg> ;purpose=icon,
  <http://www.example.com/alice/> ;purpose=info
```

20.10 Contact

A Contact header field value provides a URI whose meaning depends on the type of request or response it is in.

A Contact header field value can contain a display name, a URI with URI parameters, and header parameters.

This document defines the Contact parameters "q" and "expires". These parameters are only used when the Contact is present in a REGISTER request or response, or in a 3xx response. Additional parameters may be defined in other specifications.

When the header field value contains a display name, the URI including all URI parameters is enclosed in "<" and ">". If no "<" and ">" are present, all parameters after the URI are header parameters, not URI parameters. The display name can be tokens, or a quoted string, if a larger character set is desired.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, semicolon, or question mark. There may or may not be LWS between the display-name and the "<".

These rules for parsing a display name, URI and URI parameters, and header parameters also apply for the header fields To and From.

> The Contact header field has a role similar to the Location header field in HTTP. However, the HTTP header field only allows one address, unquoted. Since URIs can contain commas and semicolons as reserved characters, they can be mistaken for header or parameter delimiters, respectively.

The compact form of the Contact header field is m (for "moved").

Examples:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
   ;q=0.7; expires=3600,
   "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
m: <sips:bob@192.0.2.4>;expires=60
```

20.11 Content-Disposition

The Content-Disposition header field describes how the message body or, for multipart messages, a message body part is to be interpreted by the UAC or UAS. This SIP header field extends the MIME Content-Type (RFC 2183 [18]).

Several new "disposition-types" of the Content-Disposition header are defined by SIP. The value "session" indicates that the body part describes a session, for either calls or early (pre-call) media. The value "render" indicates that the body part should be displayed or otherwise rendered to the user. Note that the value "render" is used rather than "inline" to avoid the connotation that the MIME body is displayed as a part of the rendering of the entire message (since the MIME bodies of SIP messages oftentimes are not displayed to users). For backward-compatibility, if the Content-Disposition header field is missing, the server SHOULD assume bodies of Content-Type application/sdp are the disposition "session", while other content types are "render".

The disposition type "icon" indicates that the body part contains an image suitable as an iconic representation of the caller or callee that could be rendered informationally by a user agent when a message has been received, or persistently while a dialog takes place. The value "alert" indicates that the body part contains information, such as an audio clip, that should be rendered by the user agent in an attempt to alert the user to the receipt of a request, generally a request that initiates a dialog; this alerting body could for example be rendered as a ring tone for a phone call after a 180 Ringing provisional response has been sent.

Any MIME body with a "disposition-type" that renders content to the user should only be processed when a message has been properly authenticated.

The handling parameter, handling-param, describes how the UAS should react if it receives a message body whose content type or disposition type it does not understand. The parameter has defined values of "optional" and "required". If the handling parameter is missing, the value "required" SHOULD be assumed. The handling parameter is described in RFC 3204 [19].

If this header field is missing, the MIME type determines the default content disposition. If there is none, "render" is assumed.

Example:

```
Content-Disposition: session
```

20.12 Content-Encoding

The Content-Encoding header field is used as a modifier to the "media-type". When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a body to be compressed without losing the identity of its underlying media type.

If multiple encodings have been applied to an entity-body, the content codings MUST be listed in the order in which they were applied.

All content-coding values are case-insensitive. IANA acts as a registry for content-coding value tokens. See [H3.5] for a definition of the syntax for content-coding.

Clients MAY apply content encodings to the body in requests. A server MAY apply content encodings to the bodies in responses. The server MUST only use encodings listed in the Accept-Encoding header field in the request.

The compact form of the Content-Encoding header field is e. Examples:

```
Content-Encoding: gzip
e: tar
```

20.13 Content-Language

See [H14.12]. Example:

```
Content-Language: fr
```

20.14 Content-Length

The Content-Length header field indicates the size of the message-body, in decimal number of octets, sent to the recipient. Applications SHOULD use this field to indicate the size of the message-body to be transferred, regardless of the media type of the entity. If a stream-based protocol (such as TCP) is used as transport, the header field MUST be used.

The size of the message-body does not include the CRLF separating header fields and body. Any Content-Length greater than or equal to zero is a valid value. If no body is present in a message, then the Content-Length header field value MUST be set to zero.

The ability to omit Content-Length simplifies the creation of cgi-like scripts that dynamically generate responses.

The compact form of the header field is l.

Examples:

```
Content-Length: 349
l: 173
```

20.15 Content-Type

The Content-Type header field indicates the media type of the message-body sent to the recipient. The "media-type" element is defined in [H3.7]. The Content-Type header field MUST be present if the body is not empty. If the body is empty, and a Content-Type header field is present, it indicates that the body of the specific type has zero length (for example, an empty audio file).

The compact form of the header field is c.

Examples:

```
Content-Type: application/sdp
c: text/html; charset=ISO-8859-4
```

20.16 CSeq

A CSeq header field in a request contains a single decimal sequence number and the request method. The sequence number MUST be expressible as a 32-bit unsigned integer. The method part of CSeq is case-sensitive. The CSeq header field serves to order transactions within a dialog, to provide a means to uniquely identify transactions, and to differentiate between new requests and request retransmissions. Two CSeq header fields are considered equal if the sequence number and the request method are identical. Example:

```
CSeq: 4711 INVITE
```

20.17 Date

The Date header field contains the date and time. Unlike HTTP/1.1, SIP only supports the most recent RFC 1123 [20] format for dates. As in [H3.3], SIP restricts the time zone in SIP-date to "GMT", while RFC 1123 allows any time zone. An RFC 1123 date is case-sensitive.

The Date header field reflects the time when the request or response is first sent.

The Date header field can be used by simple end systems without a battery-backed clock to acquire a notion of current time. However, in its GMT form, it requires clients to know their offset from GMT.

Example:

```
Date: Sat, 13 Nov 2010 23:29:00 GMT
```

20.18 Error-Info

The Error-Info header field provides a pointer to additional information about the error status response.

SIP UACs have user interface capabilities ranging from pop-up windows and audio on PC softclients to audio-only on "black" phones or endpoints connected via gateways. Rather than forcing a server generating an error to choose between sending an error status code with a detailed reason phrase and playing an audio recording, the Error-Info header field allows both to be sent. The UAC then has the choice of which error indicator to render to the caller.

A UAC MAY treat a SIP or SIPS URI in an Error-Info header field as if it were a Contact in a redirect and generate a new INVITE, resulting in a recorded announcement session being established. A non-SIP URI MAY be rendered to the user.

Examples:

```
SIP/2.0 404 The number you have dialed is not in service
Error-Info: <sip:not-in-service-recording@atlanta.com>
```

20.19 Expires

The Expires header field gives the relative time after which the message (or content) expires.

The precise meaning of this is method dependent.

The expiration time in an INVITE does not affect the duration of the actual session that may result from the invitation. Session description protocols may offer the ability to express time limits on the session duration, however.

The value of this field is an integral number of seconds (in decimal) between 0 and (2**32)-1, measured from the receipt of the request.

Example:

```
Expires: 5
```

20.20 From

The From header field indicates the initiator of the request. This may be different from the initiator of the dialog. Requests sent by the callee to the caller use the callee's address in the From header field.

The optional "display-name" is meant to be rendered by a human user interface. A system SHOULD use the display name "Anonymous" if the identity of the client is to remain hidden. Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon. Syntax issues are discussed in Section 7.3.1.

Two From header fields are equivalent if their URIs match, and their parameters match. Extension parameters in one header field, not present in the other are ignored for the purposes of comparison. This means that the display name and presence or absence of angle brackets do not affect matching.

See Section 20.10 for the rules for parsing a display name, URI and URI parameters, and header field parameters.

The compact form of the From header field is f.

Examples:

```
From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s
From: sip:+12125551212@server.phone2net.com;tag=887s
f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

20.21 In-Reply-To

The In-Reply-To header field enumerates the Call-IDs that this call references or returns. These Call-IDs may have been cached by the client then included in this header field in a return call.

> This allows automatic call distribution systems to route return calls to the originator of the first call. This also allows callees to filter calls, so that only return calls for calls they originated will be accepted. This field is not a substitute for request authentication.

Example:

```
In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com
```

20.22 Max-Forwards

The Max-Forwards header field must be used with any SIP method to limit the number of proxies or gateways that can forward the request to the next downstream server. This can also be useful when the client is attempting to trace a request chain that appears to be failing or looping in mid-chain.

The Max-Forwards value is an integer in the range 0-255 indicating the remaining number of times this request message is allowed to be forwarded. This count is decremented by each server that forwards the request. The recommended initial value is 70.

This header field should be inserted by elements that can not otherwise guarantee loop detection. For example, a B2BUA should insert a Max-Forwards header field.

Example:

```
Max-Forwards: 6
```

20.23 Min-Expires

The Min-Expires header field conveys the minimum refresh interval supported for soft-state elements managed by that server. This includes Contact header fields that are stored by a registrar. The header field contains a decimal integer number of seconds from 0 to (2**32)-1. The use of the header field in a 423 (Interval Too Brief) response is described in Sections 10.2.8, 10.3, and 21.4.17.

Example:

```
Min-Expires: 60
```

20.24 MIME-Version

See [H19.4.1].

Example:

```
MIME-Version: 1.0
```

20.25 Organization

The Organization header field conveys the name of the organization to which the SIP element issuing the request or response belongs.

The field MAY be used by client software to filter calls.

Example:

```
Organization: Boxes by Bob
```

20.26 Priority

The Priority header field indicates the urgency of the request as perceived by the client. The Priority header field describes the priority that the SIP request should have to the receiving human or its agent. For example, it may be factored into decisions about call routing and acceptance. For these decisions, a message containing no Priority header field SHOULD be treated as if it specified a Priority of "normal". The Priority header field does not influence the use of communications resources such as packet forwarding priority in routers or access to circuits in PSTN gateways. The header field can have the values "non-urgent", "normal", "urgent", and "emergency", but additional values can be defined elsewhere. It is RECOMMENDED that the value of "emergency" only be used when life, limb, or property are in imminent danger. Otherwise, there are no semantics defined for this header field.

These are the values of RFC 2076 [38], with the addition of "emergency".

Examples:

```
Subject: A tornado is heading our way!
Priority: emergency
```

or

```
Subject: Weekend plans
Priority: non-urgent
```

20.27 Proxy-Authenticate

A Proxy-Authenticate header field value contains an authentication challenge.

The use of this header field is defined in [H14.33]. See Section 22.3 for further details on its usage.

Example:

```
Proxy-Authenticate: Digest realm="atlanta.com",
 domain="sip:ss1.carrier.com", qop="auth",
 nonce="f84f1cec41e6cbe5aea9c8e88d359",
 opaque="", stale=FALSE, algorithm=MD5
```

20.28 Proxy-Authorization

The Proxy-Authorization header field allows the client to identify itself (or its user) to a proxy that requires authentication. A Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

See Section 22.3 for a definition of the usage of this header field.

This header field, along with Authorization, breaks the general rules about multiple header field names. Although not a comma-separated list, this header field name may be present multiple times, and MUST NOT be combined into a single header line using the usual rules described in Section 7.3.1.

Example:

```
Proxy-Authorization: Digest username="Alice", realm="atlanta.com",
   nonce="c60f3082ee1212b402a21831ae",
   response="245f23415f11432b3434341c022"
```

20.29 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive features that must be supported by the proxy. See Section 20.32 for more details on the mechanics of this message and a usage example.

Example:

```
Proxy-Require: foo
```

20.30 Record-Route

The Record-Route header field is inserted by proxies in a request to force future requests in the dialog to be routed through the proxy.

Examples of its use with the Route header field are described in Sections 16.12.1.

Example:

```
Record-Route: <sip:server10.biloxi.com;lr>,
              <sip:bigbox3.site3.atlanta.com;lr>
```

20.31 Reply-To

The Reply-To header field contains a logical return URI that may be different from the From header field. For example, the URI MAY be used to return missed calls or unestablished sessions. If the user wished to remain anonymous, the header field SHOULD either be omitted from the request or populated in such a way that does not reveal any private information.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon. Syntax issues are discussed in Section 7.3.1.

Example:

```
Reply-To: Bob <sip:bob@biloxi.com>
```

20.32 Require

The Require header field is used by UACs to tell UASs about options that the UAC expects the UAS to support in order to process the request. Although an optional header field, the Require MUST NOT be ignored if it is present.

The Require header field contains a list of option tags, described in Section 19.2. Each option tag defines a SIP extension that MUST be understood to process the request. Frequently, this is used to indicate that a specific set of extension header fields need to be understood. A UAC compliant to this specification MUST only include option tags corresponding to standards-track RFCs.

Example:

```
Require: 100rel
```

20.33 Retry-After

The Retry-After header field can be used with a 500 (Server Internal Error) or 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 413 (Request Entity Too Large), 480 (Temporarily Unavailable), 486 (Busy Here), 600 (Busy), or 603

(Decline) response to indicate when the called party anticipates being available again. The value of this field is a positive integer number of seconds (in decimal) after the time of the response.

An optional comment can be used to indicate additional information about the time of callback. An optional "duration" parameter indicates how long the called party will be reachable starting at the initial time of availability. If no duration parameter is given, the service is assumed to be available indefinitely.

Examples:

```
Retry-After: 18000;duration=3600
Retry-After: 120 (I'm in a meeting)
```

20.34 Route

The Route header field is used to force routing for a request through the listed set of proxies. Examples of the use of the Route header field are in Section 16.12.1.

Example:

```
Route: <sip:bigbox3.site3.atlanta.com;lr>,
       <sip:server10.biloxi.com;lr>
```

20.35 Server

The Server header field contains information about the software used by the UAS to handle the request.

Revealing the specific software version of the server might allow the server to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make the Server header field a configurable option.

Example:

```
Server: HomeServer v2
```

20.36 Subject

The Subject header field provides a summary or indicates the nature of the call, allowing call filtering without having to parse the session description. The session description does not have to use the same subject indication as the invitation.

The compact form of the Subject header field is s.

Example:

```
Subject: Need more boxes
s: Tech Support
```

20.37 Supported

The Supported header field enumerates all the extensions supported by the UAC or UAS.

The Supported header field contains a list of option tags, described in Section 19.2, that are understood by the UAC or UAS. A UA compliant to this specification MUST only include option tags corresponding to standards-track RFCs. If empty, it means that no extensions are supported.

The compact form of the Supported header field is k.

Example:

```
Supported: 100rel
```

20.38 Timestamp

The Timestamp header field describes when the UAC sent the request to the UAS.

See Section 8.2.6 for details on how to generate a response to a request that contains the header field. Although there is no normative behavior defined here that makes use of the header, it allows for extensions or SIP applications to obtain RTT estimates.

Example:

```
Timestamp: 54
```

20.39 To

The To header field specifies the logical recipient of the request.

The optional "display-name" is meant to be rendered by a human-user interface. The "tag" parameter serves as a general mechanism for dialog identification.

See Section 19.3 for details of the "tag" parameter.

Comparison of To header fields for equality is identical to comparison of From header fields. See Section 20.10 for the rules for parsing a display name, URI and URI parameters, and header field parameters.

The compact form of the To header field is t.

The following are examples of valid To header fields:

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
t: sip:+12125551212@server.phone2net.com
```

20.40 Unsupported

The Unsupported header field lists the features not supported by the UAS. See Section 20.32 for motivation.

Example:

```
Unsupported: foo
```

20.41 User-Agent

The User-Agent header field contains information about the UAC originating the request. The semantics of this header field are defined in [H14.43].

Revealing the specific software version of the user agent might allow the user agent to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make the User-Agent header field a configurable option.

Example:

```
User-Agent: Softphone Beta1.5
```

20.42 Via

The Via header field indicates the path taken by the request so far and indicates the path that should be followed in routing responses. The branch ID parameter in the Via header field values serves as a transaction identifier, and is used by proxies to detect loops.

A Via header field value contains the transport protocol used to send the message, the client's host name or network address, and possibly the port number at which it wishes to receive responses. A Via header field value can also contain parameters such as "maddr", "ttl", "received", and "branch", whose meaning and use are described

in other sections. For implementations compliant to this specification, the value of the branch parameter MUST start with the magic cookie "z9hG4bK", as discussed in Section 8.1.1.7.

Transport protocols defined here are "UDP", "TCP", "TLS", and "SCTP". "TLS" means TLS over TCP. When a request is sent to a SIPS URI, the protocol still indicates "SIP", and the transport protocol is TLS.

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7
Via: SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207
     ;branch=z9hG4bK77asjd
```

The compact form of the Via header field is v.

In this example, the message originated from a multi-homed host with two addresses, 192.0.2.1 and 192.0.2.207. The sender guessed wrong as to which network interface would be used. Erlang.bell-telephone.com noticed the mismatch and added a parameter to the previous hop's Via header field value, containing the address that the packet actually came from.

The host or network address and port number are not required to follow the SIP URI syntax. Specifically, LWS on either side of the ":" or "/" is allowed, as shown here:

```
Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16
  ;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1
```

Even though this specification mandates that the branch parameter be present in all requests, the BNF for the header field indicates that it is optional. This allows interoperation with RFC 2543 elements, which did not have to insert the branch parameter.

Two Via header fields are equal if their sent-protocol and sent-by fields are equal, both have the same set of parameters, and the values of all parameters are equal.

20.43 Warning

The Warning header field is used to carry additional information about the status of a response. Warning header field values are sent with responses and contain a three-digit warning code, host name, and warning text.

The "warn-text" should be in a natural language that is most likely to be intelligible to the human user receiving the response. This decision can be based on any available knowledge, such as the location of the user, the Accept-Language field in a request, or the

Content-Language field in a response. The default language is i-default [21].

The currently-defined "warn-code"s are listed below, with a recommended warn-text in English and a description of their meaning. These warnings describe failures induced by the session description. The first digit of warning codes beginning with "3" indicates warnings specific to SIP. Warnings 300 through 329 are reserved for indicating problems with keywords in the session description, 330 through 339 are warnings related to basic network services requested in the session description, 370 through 379 are warnings related to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

300 Incompatible network protocol: One or more network protocols contained in the session description are not available.

301 Incompatible network address formats: One or more network address formats contained in the session description are not available.

302 Incompatible transport protocol: One or more transport protocols described in the session description are not available.

303 Incompatible bandwidth units: One or more bandwidth measurement units contained in the session description were not understood.

304 Media type not available: One or more media types contained in the session description are not available.

305 Incompatible media format: One or more media formats contained in the session description are not available.

306 Attribute not understood: One or more of the media attributes in the session description are not supported.

307 Session description parameter not understood: A parameter other than those listed above was not understood.

330 Multicast not available: The site where the user is located does not support multicast.

331 Unicast not available: The site where the user is located does not support unicast communication (usually due to the presence of a firewall).

370 Insufficient bandwidth: The bandwidth specified in the session description or defined by the media exceeds that known to be available.

399 Miscellaneous warning: The warning text can include arbitrary information to be presented to a human user or logged. A system receiving this warning MUST NOT take any automated action.

1xx and 2xx have been taken by HTTP/1.1.

Additional "warn-code"s can be defined through IANA, as defined in Section 27.2.

Examples:

```
Warning: 307 isi.edu "Session parameter 'foo' not understood"
Warning: 301 isi.edu "Incompatible network address type 'E.164'"
```

20.44 WWW-Authenticate

A WWW-Authenticate header field value contains an authentication challenge. See Section 22.2 for further details on its usage.

Example:

```
WWW-Authenticate: Digest realm="atlanta.com",
  domain="sip:boxesbybob.com", qop="auth",
  nonce="f84f1cec41e6cbe5aea9c8e88d359",
  opaque="", stale=FALSE, algorithm=MD5
```

21 Response Codes

The response codes are consistent with, and extend, HTTP/1.1 response codes. Not all HTTP/1.1 response codes are appropriate, and only those that are appropriate are given here. Other HTTP/1.1 response codes SHOULD NOT be used. Also, SIP defines a new class, 6xx.

21.1 Provisional 1xx

Provisional responses, also known as informational responses, indicate that the server contacted is performing some further action and does not yet have a definitive response. A server sends a 1xx response if it expects to take more than 200 ms to obtain a final response. Note that 1xx responses are not transmitted reliably. They never cause the client to send an ACK. Provisional (1xx) responses MAY contain message bodies, including session descriptions.

21.1.1 100 Trying

This response indicates that the request has been received by the next-hop server and that some unspecified action is being taken on behalf of this call (for example, a database is being consulted). This response, like all other provisional responses, stops retransmissions of an INVITE by a UAC. The 100 (Trying) response is different from other provisional responses, in that it is never forwarded upstream by a stateful proxy.

21.1.2 180 Ringing

The UA receiving the INVITE is trying to alert the user. This response MAY be used to initiate local ringback.

21.1.3 181 Call Is Being Forwarded

A server MAY use this status code to indicate that the call is being forwarded to a different set of destinations.

21.1.4 182 Queued

The called party is temporarily unavailable, but the server has decided to queue the call rather than reject it. When the callee becomes available, it will return the appropriate final status response. The reason phrase MAY give further details about the status of the call, for example, "5 calls queued; expected waiting time is 15 minutes". The server MAY issue several 182 (Queued) responses to update the caller about the status of the queued call.

21.1.5 183 Session Progress

The 183 (Session Progress) response is used to convey information about the progress of the call that is not otherwise classified. The Reason-Phrase, header fields, or message body MAY be used to convey more details about the call progress.

21.2 Successful 2xx

The request was successful.

21.2.1 200 OK

The request has succeeded. The information returned with the response depends on the method used in the request.

21.3 Redirection 3xx

3xx responses give information about the user's new location, or about alternative services that might be able to satisfy the call.

21.3.1 300 Multiple Choices

The address in the request resolved to several choices, each with its own specific location, and the user (or UA) can select a preferred communication end point and redirect its request to that location.

The response MAY include a message body containing a list of resource characteristics and location(s) from which the user or UA can choose the one most appropriate, if allowed by the Accept request header field. However, no MIME types have been defined for this message body.

The choices SHOULD also be listed as Contact fields (Section 20.10). Unlike HTTP, the SIP response MAY contain several Contact fields or a list of addresses in a Contact field. UAs MAY use the Contact header field value for automatic redirection or MAY ask the user to confirm a choice. However, this specification does not define any standard for such automatic selection.

> This status response is appropriate if the callee can be reached at several different locations and the server cannot or prefers not to proxy the request.

21.3.2 301 Moved Permanently

The user can no longer be found at the address in the Request-URI, and the requesting client SHOULD retry at the new address given by the Contact header field (Section 20.10). The requestor SHOULD update any local directories, address books, and user location caches with this new value and redirect future requests to the address(es) listed.

21.3.3 302 Moved Temporarily

The requesting client SHOULD retry the request at the new address(es) given by the Contact header field (Section 20.10). The Request-URI of the new request uses the value of the Contact header field in the response.

The duration of the validity of the Contact URI can be indicated through an Expires (Section 20.19) header field or an expires parameter in the Contact header field. Both proxies and UAs MAY cache this URI for the duration of the expiration time. If there is no explicit expiration time, the address is only valid once for recursing, and MUST NOT be cached for future transactions.

If the URI cached from the Contact header field fails, the Request-URI from the redirected request MAY be tried again a single time.

> The temporary URI may have become out-of-date sooner than the expiration time, and a new temporary URI may be available.

21.3.4 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Contact field. The Contact field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 (Use Proxy) responses MUST only be generated by UASs.

21.3.5 380 Alternative Service

The call was not successful, but alternative services are possible.

The alternative services are described in the message body of the response. Formats for such bodies are not defined here, and may be the subject of future standardization.

21.4 Request Failure 4xx

4xx responses are definite failure responses from a particular server. The client SHOULD NOT retry the same request without modification (for example, adding appropriate authorization). However, the same request to a different server might be successful.

21.4.1 400 Bad Request

The request could not be understood due to malformed syntax. The Reason-Phrase SHOULD identify the syntax problem in more detail, for example, "Missing Call-ID header field".

21.4.2 401 Unauthorized

The request requires user authentication. This response is issued by UASs and registrars, while 407 (Proxy Authentication Required) is used by proxy servers.

21.4.3 402 Payment Required

Reserved for future use.

21.4.4 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request SHOULD NOT be repeated.

21.4.5 404 Not Found

The server has definitive information that the user does not exist at the domain specified in the Request-URI. This status is also returned if the domain in the Request-URI does not match any of the domains handled by the recipient of the request.

21.4.6 405 Method Not Allowed

The method specified in the Request-Line is understood, but not allowed for the address identified by the Request-URI.

The response MUST include an Allow header field containing a list of valid methods for the indicated address.

21.4.7 406 Not Acceptable

The resource identified by the request is only capable of generating response entities that have content characteristics not acceptable according to the Accept header field sent in the request.

21.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with the proxy. SIP access authentication is explained in Sections 26 and 22.3.

This status code can be used for applications where access to the communication channel (for example, a telephony gateway) rather than the callee requires authentication.

21.4.9 408 Request Timeout

The server could not produce a response within a suitable amount of time, for example, if it could not determine the location of the user in time. The client MAY repeat the request without modifications at any later time.

21.4.10 410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.

21.4.11 413 Request Entity Too Large

The server is refusing to process a request because the request entity-body is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

21.4.12 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

21.4.13 415 Unsupported Media Type

The server is refusing to service the request because the message body of the request is in a format not supported by the server for the requested method. The server MUST return a list of acceptable formats using the Accept, Accept-Encoding, or Accept-Language header field, depending on the specific problem with the content. UAC processing of this response is described in Section 8.1.3.5.

21.4.14 416 Unsupported URI Scheme

The server cannot process the request because the scheme of the URI in the Request-URI is unknown to the server. Client processing of this response is described in Section 8.1.3.5.

21.4.15 420 Bad Extension

The server did not understand the protocol extension specified in a Proxy-Require (Section 20.29) or Require (Section 20.32) header field. The server MUST include a list of the unsupported extensions in an Unsupported header field in the response. UAC processing of this response is described in Section 8.1.3.5.

21.4.16 421 Extension Required

The UAS needs a particular extension to process the request, but this extension is not listed in a Supported header field in the request. Responses with this status code MUST contain a Require header field listing the required extensions.

A UAS SHOULD NOT use this response unless it truly cannot provide any useful service to the client. Instead, if a desirable extension is not listed in the Supported header field, servers SHOULD process the request using baseline SIP capabilities and any extensions supported by the client.

21.4.17 423 Interval Too Brief

The server is rejecting the request because the expiration time of the resource refreshed by the request is too short. This response can be used by a registrar to reject a registration whose Contact header field expiration time was too small. The use of this response and the related Min-Expires header field are described in Sections 10.2.8, 10.3, and 20.23.

21.4.18 480 Temporarily Unavailable

The callee's end system was contacted successfully but the callee is currently unavailable (for example, is not logged in, logged in but in a state that precludes communication with the callee, or has activated the "do not disturb" feature). The response MAY indicate a better time to call in the Retry-After header field. The user could also be available elsewhere (unbeknownst to this server). The reason phrase SHOULD indicate a more precise cause as to why the callee is unavailable. This value SHOULD be settable by the UA. Status 486 (Busy Here) MAY be used to more precisely indicate a particular reason for the call failure.

This status is also returned by a redirect or proxy server that recognizes the user identified by the Request-URI, but does not currently have a valid forwarding location for that user.

21.4.19 481 Call/Transaction Does Not Exist

This status indicates that the UAS received a request that does not match any existing dialog or transaction.

21.4.20 482 Loop Detected

The server has detected a loop (Section 16.3 Item 4).

21.4.21 483 Too Many Hops

The server received a request that contains a Max-Forwards (Section 20.22) header field with the value zero.

21.4.22 484 Address Incomplete

The server received a request with a Request-URI that was incomplete. Additional information SHOULD be provided in the reason phrase.

This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a 484 (Address Incomplete) status response.

21.4.23 485 Ambiguous

The Request-URI was ambiguous. The response MAY contain a listing of possible unambiguous addresses in Contact header fields. Revealing alternatives can infringe on privacy of the user or the organization. It MUST be possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of possible choices for ambiguous Request-URIs.

Example response to a request with the Request-URI sip:lee@example.com:

```
SIP/2.0 485 Ambiguous
Contact: Carol Lee <sip:carol.lee@example.com>
Contact: Ping Lee <sip:p.lee@example.com>
Contact: Lee M. Foote <sips:lee.foote@example.com>
```

Some email and voice mail systems provide this functionality. A status code separate from 3xx is used since the semantics are different: for 300, it is assumed that the same person or service will be reached by the choices provided. While an automated choice or sequential search makes sense for a 3xx response, user intervention is required for a 485 (Ambiguous) response.

21.4.24 486 Busy Here

The callee’s end system was contacted successfully, but the callee is currently not willing or able to take additional calls at this end system. The response MAY indicate a better time to call in the Retry-After header field. The user could also be available

elsewhere, such as through a voice mail service. Status 600 (Busy Everywhere) SHOULD be used if the client knows that no other end system will be able to accept this call.

21.4.25 487 Request Terminated

The request was terminated by a BYE or CANCEL request. This response is never returned for a CANCEL request itself.

21.4.26 488 Not Acceptable Here

The response has the same meaning as 606 (Not Acceptable), but only applies to the specific resource addressed by the Request-URI and the request may succeed elsewhere.

A message body containing a description of media capabilities MAY be present in the response, which is formatted according to the Accept header field in the INVITE (or application/sdp if not present), the same as a message body in a 200 (OK) response to an OPTIONS request.

21.4.27 491 Request Pending

The request was received by a UAS that had a pending request within the same dialog. Section 14.2 describes how such "glare" situations are resolved.

21.4.28 493 Undecipherable

The request was received by a UAS that contained an encrypted MIME body for which the recipient does not possess or will not provide an appropriate decryption key. This response MAY have a single body containing an appropriate public key that should be used to encrypt MIME bodies sent to this UA. Details of the usage of this response code can be found in Section 23.2.

21.5 Server Failure 5xx

5xx responses are failure responses given when a server itself has erred.

21.5.1 500 Server Internal Error

The server encountered an unexpected condition that prevented it from fulfilling the request. The client MAY display the specific error condition and MAY retry the request after several seconds.

If the condition is temporary, the server MAY indicate when the client may retry the request using the Retry-After header field.

21.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when a UAS does not recognize the request method and is not capable of supporting it for any user. (Proxies forward all requests regardless of method.)

Note that a 405 (Method Not Allowed) is sent when the server recognizes the request method, but that method is not allowed or supported.

21.5.3 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfill the request.

21.5.4 503 Service Unavailable

The server is temporarily unable to process the request due to a temporary overloading or maintenance of the server. The server MAY indicate when the client should retry the request in a Retry-After header field. If no Retry-After is given, the client MUST act as if it had received a 500 (Server Internal Error) response.

A client (proxy or UAC) receiving a 503 (Service Unavailable) SHOULD attempt to forward the request to an alternate server. It SHOULD NOT forward any other requests to that server for the duration specified in the Retry-After header field, if present.

Servers MAY refuse the connection or drop the request instead of responding with 503 (Service Unavailable).

21.5.5 504 Server Time-out

The server did not receive a timely response from an external server it accessed in attempting to process the request. 408 (Request Timeout) should be used instead if there was no response within the period specified in the Expires header field from the upstream server.

21.5.6 505 Version Not Supported

The server does not support, or refuses to support, the SIP protocol version that was used in the request. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message.

21.5.7 513 Message Too Large

The server was unable to process the request since the message length exceeded its capabilities.

21.6 Global Failures 6xx

6xx responses indicate that a server has definitive information about a particular user, not just the particular instance indicated in the Request-URI.

21.6.1 600 Busy Everywhere

The callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time. The response MAY indicate a better time to call in the Retry-After header field. If the callee does not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead. This status response is returned only if the client knows that no other end point (such as a voice mail system) will answer the request. Otherwise, 486 (Busy Here) should be returned.

21.6.2 603 Decline

The callee's machine was successfully contacted but the user explicitly does not wish to or cannot participate. The response MAY indicate a better time to call in the Retry-After header field. This status response is returned only if the client knows that no other end point will answer the request.

21.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in the Request-URI does not exist anywhere.

21.6.4 606 Not Acceptable

The user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable.

A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately support the session described. The 606 (Not Acceptable) response MAY contain a list of reasons in a Warning header field describing why the session described cannot be supported. Warning reason codes are listed in Section 20.43.

A message body containing a description of media capabilities MAY be present in the response, which is formatted according to the Accept header field in the INVITE (or application/sdp if not present), the same as a message body in a 200 (OK) response to an OPTIONS request.

It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide whether or not to act on a 606 (Not Acceptable) response.

This status response is returned only if the client knows that no other end point will answer the request.

22 Usage of HTTP Authentication

SIP provides a stateless, challenge-based mechanism for authentication that is based on authentication in HTTP. Any time that a proxy server or UA receives a request (with the exceptions given in Section 22.1), it MAY challenge the initiator of the request to provide assurance of its identity. Once the originator has been identified, the recipient of the request SHOULD ascertain whether or not this user is authorized to make the request in question. No authorization systems are recommended or discussed in this document.

The "Digest" authentication mechanism described in this section provides message authentication and replay protection only, without message integrity or confidentiality. Protective measures above and beyond those provided by Digest need to be taken to prevent active attackers from modifying SIP requests and responses.

Note that due to its weak security, the usage of "Basic" authentication has been deprecated. Servers MUST NOT accept credentials using the "Basic" authorization scheme, and servers also MUST NOT challenge with "Basic". This is a change from RFC 2543.

22.1 Framework

The framework for SIP authentication closely parallels that of HTTP (RFC 2617 [17]). In particular, the BNF for auth-scheme, auth-param, challenge, realm, realm-value, and credentials is identical (although the usage of "Basic" as a scheme is not permitted). In SIP, a UAS uses the 401 (Unauthorized) response to challenge the identity of a UAC. Additionally, registrars and redirect servers MAY make use of 401 (Unauthorized) responses for authentication, but proxies MUST NOT, and instead MAY use the 407 (Proxy Authentication Required)

response. The requirements for inclusion of the Proxy-Authenticate, Proxy-Authorization, WWW-Authenticate, and Authorization in the various messages are identical to those described in RFC 2617 [17].

Since SIP does not have the concept of a canonical root URL, the notion of protection spaces is interpreted differently in SIP. The realm string alone defines the protection domain. This is a change from RFC 2543, in which the Request-URI and the realm together defined the protection domain.

> This previous definition of protection domain caused some amount of confusion since the Request-URI sent by the UAC and the Request-URI received by the challenging server might be different, and indeed the final form of the Request-URI might not be known to the UAC. Also, the previous definition depended on the presence of a SIP URI in the Request-URI and seemed to rule out alternative URI schemes (for example, the tel URL).

Operators of user agents or proxy servers that will authenticate received requests MUST adhere to the following guidelines for creation of a realm string for their server:

- o Realm strings MUST be globally unique. It is RECOMMENDED that a realm string contain a hostname or domain name, following the recommendation in Section 3.2.1 of RFC 2617 [17].

- o Realm strings SHOULD present a human-readable identifier that can be rendered to a user.

For example:

```
INVITE sip:bob@biloxi.com SIP/2.0
Authorization: Digest realm="biloxi.com", <...>
```

Generally, SIP authentication is meaningful for a specific realm, a protection domain. Thus, for Digest authentication, each such protection domain has its own set of usernames and passwords. If a server does not require authentication for a particular request, it MAY accept a default username, "anonymous", which has no password (password of ""). Similarly, UACs representing many users, such as PSTN gateways, MAY have their own device-specific username and password, rather than accounts for particular users, for their realm.

While a server can legitimately challenge most SIP requests, there are two requests defined by this document that require special handling for authentication: ACK and CANCEL.

Under an authentication scheme that uses responses to carry values used to compute nonces (such as Digest), some problems come up for any requests that take no response, including ACK. For this reason, any credentials in the INVITE that were accepted by a server MUST be accepted by that server for the ACK. UACs creating an ACK message will duplicate all of the Authorization and Proxy-Authorization header field values that appeared in the INVITE to which the ACK corresponds. Servers MUST NOT attempt to challenge an ACK.

Although the CANCEL method does take a response (a 2xx), servers MUST NOT attempt to challenge CANCEL requests since these requests cannot be resubmitted. Generally, a CANCEL request SHOULD be accepted by a server if it comes from the same hop that sent the request being canceled (provided that some sort of transport or network layer security association, as described in Section 26.2.1, is in place).

When a UAC receives a challenge, it SHOULD render to the user the contents of the "realm" parameter in the challenge (which appears in either a WWW-Authenticate header field or Proxy-Authenticate header field) if the UAC device does not already know of a credential for the realm in question. A service provider that pre-configures UAs with credentials for its realm should be aware that users will not have the opportunity to present their own credentials for this realm when challenged at a pre-configured device.

Finally, note that even if a UAC can locate credentials that are associated with the proper realm, the potential exists that these credentials may no longer be valid or that the challenging server will not accept these credentials for whatever reason (especially when "anonymous" with no password is submitted). In this instance a server may repeat its challenge, or it may respond with a 403 Forbidden. A UAC MUST NOT re-attempt requests with the credentials that have just been rejected (though the request may be retried if the nonce was stale).

22.2 User-to-User Authentication

When a UAS receives a request from a UAC, the UAS MAY authenticate the originator before the request is processed. If no credentials (in the Authorization header field) are provided in the request, the UAS can challenge the originator to provide credentials by rejecting the request with a 401 (Unauthorized) status code.

The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the realm.

An example of the WWW-Authenticate header field in a 401 challenge is:

```
WWW-Authenticate: Digest
        realm="biloxi.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

When the originating UAC receives the 401 (Unauthorized), it SHOULD, if it is able, re-originate the request with the proper credentials. The UAC may require input from the originating user before proceeding. Once authentication credentials have been supplied (either directly by the user, or discovered in an internal keyring), UAs SHOULD cache the credentials for a given value of the To header field and "realm" and attempt to re-use these values on the next request for that destination. UAs MAY cache credentials in any way they would like.

If no credentials for a realm can be located, UACs MAY attempt to retry the request with a username of "anonymous" and no password (a password of "").

Once credentials have been located, any UA that wishes to authenticate itself with a UAS or registrar -- usually, but not necessarily, after receiving a 401 (Unauthorized) response -- MAY do so by including an Authorization header field with the request. The Authorization field value consists of credentials containing the authentication information of the UA for the realm of the resource being requested as well as parameters required in support of authentication and replay protection.

An example of the Authorization header field is:

```
Authorization: Digest username="bob",
        realm="biloxi.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="sip:bob@biloxi.com",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

When a UAC resubmits a request with its credentials after receiving a 401 (Unauthorized) or 407 (Proxy Authentication Required) response, it MUST increment the CSeq header field value as it would normally when sending an updated request.

22.3 Proxy-to-User Authentication

Similarly, when a UAC sends a request to a proxy server, the proxy server MAY authenticate the originator before the request is processed. If no credentials (in the Proxy-Authorization header field) are provided in the request, the proxy can challenge the originator to provide credentials by rejecting the request with a 407 (Proxy Authentication Required) status code. The proxy MUST populate the 407 (Proxy Authentication Required) message with a Proxy-Authenticate header field value applicable to the proxy for the requested resource.

The use of Proxy-Authenticate and Proxy-Authorization parallel that described in [17], with one difference. Proxies MUST NOT add values to the Proxy-Authorization header field. All 407 (Proxy Authentication Required) responses MUST be forwarded upstream toward the UAC following the procedures for any other response. It is the UAC's responsibility to add the Proxy-Authorization header field value containing credentials for the realm of the proxy that has asked for authentication.

> If a proxy were to resubmit a request adding a Proxy-Authorization header field value, it would need to increment the CSeq in the new request. However, this would cause the UAC that submitted the original request to discard a response from the UAS, as the CSeq value would be different.

When the originating UAC receives the 407 (Proxy Authentication Required) it SHOULD, if it is able, re-originate the request with the proper credentials. It should follow the same procedures for the display of the "realm" parameter that are given above for responding to 401.

If no credentials for a realm can be located, UACs MAY attempt to retry the request with a username of "anonymous" and no password (a password of "").

The UAC SHOULD also cache the credentials used in the re-originated request.

The following rule is RECOMMENDED for proxy credential caching:

If a UA receives a Proxy-Authenticate header field value in a 401/407 response to a request with a particular Call-ID, it should incorporate credentials for that realm in all subsequent requests that contain the same Call-ID. These credentials MUST NOT be cached across dialogs; however, if a UA is configured with the realm of its local outbound proxy, when one exists, then the UA MAY cache

credentials for that realm across dialogs. Note that this does mean a future request in a dialog could contain credentials that are not needed by any proxy along the Route header path.

Any UA that wishes to authenticate itself to a proxy server -- usually, but not necessarily, after receiving a 407 (Proxy Authentication Required) response -- MAY do so by including a Proxy-Authorization header field value with the request. The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy that requires authentication. The Proxy-Authorization header field value consists of credentials containing the authentication information of the UA for the proxy and/or realm of the resource being requested.

A Proxy-Authorization header field value applies only to the proxy whose realm is identified in the "realm" parameter (this proxy may previously have demanded authentication using the Proxy-Authenticate field). When multiple proxies are used in a chain, a Proxy-Authorization header field value MUST NOT be consumed by any proxy whose realm does not match the "realm" parameter specified in that value.

Note that if an authentication scheme that does not support realms is used in the Proxy-Authorization header field, a proxy server MUST attempt to parse all Proxy-Authorization header field values to determine whether one of them has what the proxy server considers to be valid credentials. Because this is potentially very time-consuming in large networks, proxy servers SHOULD use an authentication scheme that supports realms in the Proxy-Authorization header field.

If a request is forked (as described in Section 16.7), various proxy servers and/or UAs may wish to challenge the UAC. In this case, the forking proxy server is responsible for aggregating these challenges into a single response. Each WWW-Authenticate and Proxy-Authenticate value received in responses to the forked request MUST be placed into the single response that is sent by the forking proxy to the UA; the ordering of these header field values is not significant.

> When a proxy server issues a challenge in response to a request, it will not proxy the request until the UAC has retried the request with valid credentials. A forking proxy may forward a request simultaneously to multiple proxy servers that require authentication, each of which in turn will not forward the request until the originating UAC has authenticated itself in their respective realm. If the UAC does not provide credentials for

each challenge, the proxy servers that issued the challenges will not forward requests to the UA where the destination user might be located, and therefore, the virtues of forking are largely lost.

When resubmitting its request in response to a 401 (Unauthorized) or 407 (Proxy Authentication Required) that contains multiple challenges, a UAC MAY include an Authorization value for each WWW-Authenticate value and a Proxy-Authorization value for each Proxy-Authenticate value for which the UAC wishes to supply a credential. As noted above, multiple credentials in a request SHOULD be differentiated by the "realm" parameter.

It is possible for multiple challenges associated with the same realm to appear in the same 401 (Unauthorized) or 407 (Proxy Authentication Required). This can occur, for example, when multiple proxies within the same administrative domain, which use a common realm, are reached by a forking request. When it retries a request, a UAC MAY therefore supply multiple credentials in Authorization or Proxy-Authorization header fields with the same "realm" parameter value. The same credentials SHOULD be used for the same realm.

22.4 The Digest Authentication Scheme

This section describes the modifications and clarifications required to apply the HTTP Digest authentication scheme to SIP. The SIP scheme usage is almost completely identical to that for HTTP [17].

Since RFC 2543 is based on HTTP Digest as defined in RFC 2069 [39], SIP servers supporting RFC 2617 MUST ensure they are backwards compatible with RFC 2069. Procedures for this backwards compatibility are specified in RFC 2617. Note, however, that SIP servers MUST NOT accept or request Basic authentication.

The rules for Digest authentication follow those defined in [17], with "HTTP/1.1" replaced by "SIP/2.0" in addition to the following differences:

1. The URI included in the challenge has the following BNF:

   URI = SIP-URI / SIPS-URI

2. The BNF in RFC 2617 has an error in that the 'uri' parameter of the Authorization header field for HTTP Digest

authentication is not enclosed in quotation marks. (The example in Section 3.5 of RFC 2617 is correct.) For SIP, the 'uri' MUST be enclosed in quotation marks.

3. The BNF for digest-uri-value is:

```
digest-uri-value  =  Request-URI ; as defined in Section 25
```

4. The example procedure for choosing a nonce based on Etag does not work for SIP.

5. The text in RFC 2617 [17] regarding cache operation does not apply to SIP.

6. RFC 2617 [17] requires that a server check that the URI in the request line and the URI included in the Authorization header field point to the same resource. In a SIP context, these two URIs may refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the Request-URI in the Authorization header field value corresponds to a user for whom the server is willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are not equivalent.

7. As a clarification to the calculation of the A2 value for message integrity assurance in the Digest authentication scheme, implementers should assume, when the entity-body is empty (that is, when SIP messages have no body) that the hash of the entity-body resolves to the MD5 hash of an empty string, or:

```
   H(entity-body) = MD5("") =
"d41d8cd98f00b204e9800998ecf8427e"
```

8. RFC 2617 notes that a cnonce value MUST NOT be sent in an Authorization (and by extension Proxy-Authorization) header field if no qop directive has been sent. Therefore, any algorithms that have a dependency on the cnonce (including "MD5-Sess") require that the qop directive be sent. Use of the "qop" parameter is optional in RFC 2617 for the purposes of backwards compatibility with RFC 2069; since RFC 2543 was based on RFC 2069, the "qop" parameter must unfortunately remain optional for clients and servers to receive. However, servers MUST always send a "qop" parameter in WWW-Authenticate and Proxy-Authenticate header field values. If a client receives a "qop" parameter in a challenge header field, it MUST send the "qop" parameter in any resulting authorization header field.

RFC 2543 did not allow usage of the Authentication-Info header field (it effectively used RFC 2069). However, we now allow usage of this header field, since it provides integrity checks over the bodies and provides mutual authentication. RFC 2617 [17] defines mechanisms for backwards compatibility using the qop attribute in the request. These mechanisms MUST be used by a server to determine if the client supports the new mechanisms in RFC 2617 that were not specified in RFC 2069.

23 S/MIME

SIP messages carry MIME bodies and the MIME standard includes mechanisms for securing MIME contents to ensure both integrity and confidentiality (including the 'multipart/signed' and 'application/pkcs7-mime' MIME types, see RFC 1847 [22], RFC 2630 [23] and RFC 2633 [24]). Implementers should note, however, that there may be rare network intermediaries (not typical proxy servers) that rely on viewing or modifying the bodies of SIP messages (especially SDP), and that secure MIME may prevent these sorts of intermediaries from functioning.

This applies particularly to certain types of firewalls.

The PGP mechanism for encrypting the header fields and bodies of SIP messages described in RFC 2543 has been deprecated.

23.1 S/MIME Certificates

The certificates that are used to identify an end-user for the purposes of S/MIME differ from those used by servers in one important respect - rather than asserting that the identity of the holder corresponds to a particular hostname, these certificates assert that the holder is identified by an end-user address. This address is composed of the concatenation of the "userinfo" "@" and "domainname" portions of a SIP or SIPS URI (in other words, an email address of the form "bob@biloxi.com"), most commonly corresponding to a user's address-of-record.

These certificates are also associated with keys that are used to sign or encrypt bodies of SIP messages. Bodies are signed with the private key of the sender (who may include their public key with the message as appropriate), but bodies are encrypted with the public key of the intended recipient. Obviously, senders must have foreknowledge of the public key of recipients in order to encrypt message bodies. Public keys can be stored within a UA on a virtual keyring.

Each user agent that supports S/MIME MUST contain a keyring specifically for end-users' certificates. This keyring should map between addresses of record and corresponding certificates. Over time, users SHOULD use the same certificate when they populate the originating URI of signaling (the From header field) with the same address-of-record.

Any mechanisms depending on the existence of end-user certificates are seriously limited in that there is virtually no consolidated authority today that provides certificates for end-user applications. However, users SHOULD acquire certificates from known public certificate authorities. As an alternative, users MAY create self-signed certificates. The implications of self-signed certificates are explored further in Section 26.4.2. Implementations may also use pre-configured certificates in deployments in which a previous trust relationship exists between all SIP entities.

Above and beyond the problem of acquiring an end-user certificate, there are few well-known centralized directories that distribute end-user certificates. However, the holder of a certificate SHOULD publish their certificate in any public directories as appropriate. Similarly, UACs SHOULD support a mechanism for importing (manually or automatically) certificates discovered in public directories corresponding to the target URIs of SIP requests.

23.2 S/MIME Key Exchange

SIP itself can also be used as a means to distribute public keys in the following manner.

Whenever the CMS SignedData message is used in S/MIME for SIP, it MUST contain the certificate bearing the public key necessary to verify the signature.

When a UAC sends a request containing an S/MIME body that initiates a dialog, or sends a non-INVITE request outside the context of a dialog, the UAC SHOULD structure the body as an S/MIME 'multipart/signed' CMS SignedData body. If the desired CMS service is EnvelopedData (and the public key of the target user is known), the UAC SHOULD send the EnvelopedData message encapsulated within a SignedData message.

When a UAS receives a request containing an S/MIME CMS body that includes a certificate, the UAS SHOULD first validate the certificate, if possible, with any available root certificates for certificate authorities. The UAS SHOULD also determine the subject of the certificate (for S/MIME, the SubjectAltName will contain the appropriate identity) and compare this value to the From header field

of the request. If the certificate cannot be verified, because it is self-signed, or signed by no known authority, or if it is verifiable but its subject does not correspond to the From header field of request, the UAS MUST notify its user of the status of the certificate (including the subject of the certificate, its signer, and any key fingerprint information) and request explicit permission before proceeding. If the certificate was successfully verified and the subject of the certificate corresponds to the From header field of the SIP request, or if the user (after notification) explicitly authorizes the use of the certificate, the UAS SHOULD add this certificate to a local keyring, indexed by the address-of-record of the holder of the certificate.

When a UAS sends a response containing an S/MIME body that answers the first request in a dialog, or a response to a non-INVITE request outside the context of a dialog, the UAS SHOULD structure the body as an S/MIME 'multipart/signed' CMS SignedData body. If the desired CMS service is EnvelopedData, the UAS SHOULD send the EnvelopedData message encapsulated within a SignedData message.

When a UAC receives a response containing an S/MIME CMS body that includes a certificate, the UAC SHOULD first validate the certificate, if possible, with any appropriate root certificate. The UAC SHOULD also determine the subject of the certificate and compare this value to the To field of the response; although the two may very well be different, and this is not necessarily indicative of a security breach. If the certificate cannot be verified because it is self-signed, or signed by no known authority, the UAC MUST notify its user of the status of the certificate (including the subject of the certificate, its signator, and any key fingerprint information) and request explicit permission before proceeding. If the certificate was successfully verified, and the subject of the certificate corresponds to the To header field in the response, or if the user (after notification) explicitly authorizes the use of the certificate, the UAC SHOULD add this certificate to a local keyring, indexed by the address-of-record of the holder of the certificate. If the UAC had not transmitted its own certificate to the UAS in any previous transaction, it SHOULD use a CMS SignedData body for its next request or response.

On future occasions, when the UA receives requests or responses that contain a From header field corresponding to a value in its keyring, the UA SHOULD compare the certificate offered in these messages with the existing certificate in its keyring. If there is a discrepancy, the UA MUST notify its user of a change of the certificate (preferably in terms that indicate that this is a potential security breach) and acquire the user's permission before continuing to

process the signaling. If the user authorizes this certificate, it SHOULD be added to the keyring alongside any previous value(s) for this address-of-record.

Note well however, that this key exchange mechanism does not guarantee the secure exchange of keys when self-signed certificates, or certificates signed by an obscure authority, are used - it is vulnerable to well-known attacks. In the opinion of the authors, however, the security it provides is proverbially better than nothing; it is in fact comparable to the widely used SSH application. These limitations are explored in greater detail in Section 26.4.2.

If a UA receives an S/MIME body that has been encrypted with a public key unknown to the recipient, it MUST reject the request with a 493 (Undecipherable) response. This response SHOULD contain a valid certificate for the respondent (corresponding, if possible, to any address of record given in the To header field of the rejected request) within a MIME body with a 'certs-only' "smime-type" parameter.

A 493 (Undecipherable) sent without any certificate indicates that the respondent cannot or will not utilize S/MIME encrypted messages, though they may still support S/MIME signatures.

Note that a user agent that receives a request containing an S/MIME body that is not optional (with a Content-Disposition header "handling" parameter of "required") MUST reject the request with a 415 Unsupported Media Type response if the MIME type is not understood. A user agent that receives such a response when S/MIME is sent SHOULD notify its user that the remote device does not support S/MIME, and it MAY subsequently resend the request without S/MIME, if appropriate; however, this 415 response may constitute a downgrade attack.

If a user agent sends an S/MIME body in a request, but receives a response that contains a MIME body that is not secured, the UAC SHOULD notify its user that the session could not be secured. However, if a user agent that supports S/MIME receives a request with an unsecured body, it SHOULD NOT respond with a secured body, but if it expects S/MIME from the sender (for example, because the sender's From header field value corresponds to an identity on its keychain), the UAS SHOULD notify its user that the session could not be secured.

A number of conditions that arise in the previous text call for the notification of the user when an anomalous certificate-management event occurs. Users might well ask what they should do under these circumstances. First and foremost, an unexpected change in a certificate, or an absence of security when security is expected, are

causes for caution but not necessarily indications that an attack is in progress. Users might abort any connection attempt or refuse a connection request they have received; in telephony parlance, they could hang up and call back. Users may wish to find an alternate means to contact the other party and confirm that their key has legitimately changed. Note that users are sometimes compelled to change their certificates, for example when they suspect that the secrecy of their private key has been compromised. When their private key is no longer private, users must legitimately generate a new key and re-establish trust with any users that held their old key.

Finally, if during the course of a dialog a UA receives a certificate in a CMS SignedData message that does not correspond with the certificates previously exchanged during a dialog, the UA MUST notify its user of the change, preferably in terms that indicate that this is a potential security breach.

23.3 Securing MIME bodies

There are two types of secure MIME bodies that are of interest to SIP: use of these bodies should follow the S/MIME specification [24] with a few variations.

- "multipart/signed" MUST be used only with CMS detached signatures.

  This allows backwards compatibility with non-S/MIME-compliant recipients.

- S/MIME bodies SHOULD have a Content-Disposition header field, and the value of the "handling" parameter SHOULD be "required."

- If a UAC has no certificate on its keyring associated with the address-of-record to which it wants to send a request, it cannot send an encrypted "application/pkcs7-mime" MIME message. UACs MAY send an initial request such as an OPTIONS message with a CMS detached signature in order to solicit the certificate of the remote side (the signature SHOULD be over a "message/sip" body of the type described in Section 23.4).

  Note that future standardization work on S/MIME may define non-certificate based keys.

- Senders of S/MIME bodies SHOULD use the "SMIMECapabilities" (see Section 2.5.2 of [24]) attribute to express their capabilities and preferences for further communications. Note especially that senders MAY use the "preferSignedData"

capability to encourage receivers to respond with CMS SignedData messages (for example, when sending an OPTIONS request as described above).

- S/MIME implementations MUST at a minimum support SHA1 as a digital signature algorithm, and 3DES as an encryption algorithm. All other signature and encryption algorithms MAY be supported. Implementations can negotiate support for these algorithms with the "SMIMECapabilities" attribute.

- Each S/MIME body in a SIP message SHOULD be signed with only one certificate. If a UA receives a message with multiple signatures, the outermost signature should be treated as the single certificate for this body. Parallel signatures SHOULD NOT be used.

  The following is an example of an encrypted S/MIME SDP body within a SIP message:

```
  INVITE sip:bob@biloxi.com SIP/2.0
  Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
  To: Bob <sip:bob@biloxi.com>
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710
  CSeq: 314159 INVITE
  Max-Forwards: 70
  Contact: <sip:alice@pc33.atlanta.com>
  Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
       name=smime.p7m
  Content-Disposition: attachment; filename=smime.p7m
     handling=required

*******************************************************
* Content-Type: application/sdp                       *
*                                                     *
* v=0                                                 *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com *
* s=-                                                 *
* t=0 0                                               *
* c=IN IP4 pc33.atlanta.com                           *
* m=audio 3456 RTP/AVP 0 1 3 99                       *
* a=rtpmap:0 PCMU/8000                                *
*******************************************************
```

23.4 SIP Header Privacy and Integrity using S/MIME: Tunneling SIP

As a means of providing some degree of end-to-end authentication, integrity or confidentiality for SIP header fields, S/MIME can encapsulate entire SIP messages within MIME bodies of type "message/sip" and then apply MIME security to these bodies in the same manner as typical SIP bodies. These encapsulated SIP requests and responses do not constitute a separate dialog or transaction, they are a copy of the "outer" message that is used to verify integrity or to supply additional information.

If a UAS receives a request that contains a tunneled "message/sip" S/MIME body, it SHOULD include a tunneled "message/sip" body in the response with the same smime-type.

Any traditional MIME bodies (such as SDP) SHOULD be attached to the "inner" message so that they can also benefit from S/MIME security. Note that "message/sip" bodies can be sent as a part of a MIME "multipart/mixed" body if any unsecured MIME types should also be transmitted in a request.

23.4.1 Integrity and Confidentiality Properties of SIP Headers

When the S/MIME integrity or confidentiality mechanisms are used, there may be discrepancies between the values in the "inner" message and values in the "outer" message. The rules for handling any such differences for all of the header fields described in this document are given in this section.

Note that for the purposes of loose timestamping, all SIP messages that tunnel "message/sip" SHOULD contain a Date header in both the "inner" and "outer" headers.

23.4.1.1 Integrity

Whenever integrity checks are performed, the integrity of a header field should be determined by matching the value of the header field in the signed body with that in the "outer" messages using the comparison rules of SIP as described in 20.

Header fields that can be legitimately modified by proxy servers are: Request-URI, Via, Record-Route, Route, Max-Forwards, and Proxy-Authorization. If these header fields are not intact end-to-end, implementations SHOULD NOT consider this a breach of security. Changes to any other header fields defined in this document constitute an integrity violation; users MUST be notified of a discrepancy.

23.4.1.2 Confidentiality

When messages are encrypted, header fields may be included in the encrypted body that are not present in the "outer" message.

Some header fields must always have a plaintext version because they are required header fields in requests and responses - these include:

To, From, Call-ID, CSeq, Contact. While it is probably not useful to provide an encrypted alternative for the Call-ID, CSeq, or Contact, providing an alternative to the information in the "outer" To or From is permitted. Note that the values in an encrypted body are not used for the purposes of identifying transactions or dialogs - they are merely informational. If the From header field in an encrypted body differs from the value in the "outer" message, the value within the encrypted body SHOULD be displayed to the user, but MUST NOT be used in the "outer" header fields of any future messages.

Primarily, a user agent will want to encrypt header fields that have an end-to-end semantic, including: Subject, Reply-To, Organization, Accept, Accept-Encoding, Accept-Language, Alert-Info, Error-Info, Authentication-Info, Expires, In-Reply-To, Require, Supported, Unsupported, Retry-After, User-Agent, Server, and Warning. If any of these header fields are present in an encrypted body, they should be used instead of any "outer" header fields, whether this entails displaying the header field values to users or setting internal states in the UA. They SHOULD NOT however be used in the "outer" headers of any future messages.

If present, the Date header field MUST always be the same in the "inner" and "outer" headers.

Since MIME bodies are attached to the "inner" message, implementations will usually encrypt MIME-specific header fields, including: MIME-Version, Content-Type, Content-Length, Content-Language, Content-Encoding and Content-Disposition. The "outer" message will have the proper MIME header fields for S/MIME bodies. These header fields (and any MIME bodies they preface) should be treated as normal MIME header fields and bodies received in a SIP message.

It is not particularly useful to encrypt the following header fields: Min-Expires, Timestamp, Authorization, Priority, and WWW-Authenticate. This category also includes those header fields that can be changed by proxy servers (described in the preceding section). UAs SHOULD never include these in an "inner" message if they are not

included in the "outer" message. UAs that receive any of these header fields in an encrypted body SHOULD ignore the encrypted values.

Note that extensions to SIP may define additional header fields; the authors of these extensions should describe the integrity and confidentiality properties of such header fields. If a SIP UA encounters an unknown header field with an integrity violation, it MUST ignore the header field.

23.4.2 Tunneling Integrity and Authentication

Tunneling SIP messages within S/MIME bodies can provide integrity for SIP header fields if the header fields that the sender wishes to secure are replicated in a "message/sip" MIME body signed with a CMS detached signature.

Provided that the "message/sip" body contains at least the fundamental dialog identifiers (To, From, Call-ID, CSeq), then a signed MIME body can provide limited authentication. At the very least, if the certificate used to sign the body is unknown to the recipient and cannot be verified, the signature can be used to ascertain that a later request in a dialog was transmitted by the same certificate-holder that initiated the dialog. If the recipient of the signed MIME body has some stronger incentive to trust the certificate (they were able to validate it, they acquired it from a trusted repository, or they have used it frequently) then the signature can be taken as a stronger assertion of the identity of the subject of the certificate.

In order to eliminate possible confusions about the addition or subtraction of entire header fields, senders SHOULD replicate all header fields from the request within the signed body. Any message bodies that require integrity protection MUST be attached to the "inner" message.

If a Date header is present in a message with a signed body, the recipient SHOULD compare the header field value with its own internal clock, if applicable. If a significant time discrepancy is detected (on the order of an hour or more), the user agent SHOULD alert the user to the anomaly, and note that it is a potential security breach.

If an integrity violation in a message is detected by its recipient, the message MAY be rejected with a 403 (Forbidden) response if it is a request, or any existing dialog MAY be terminated. UAs SHOULD notify users of this circumstance and request explicit guidance on how to proceed.

The following is an example of the use of a tunneled "message/sip" body:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
   handling=required
```

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-

23.4.3 Tunneling Encryption

It may also be desirable to use this mechanism to encrypt a "message/sip" MIME body within a CMS EnvelopedData message S/MIME body, but in practice, most header fields are of at least some use to the network; the general use of encryption with S/MIME is to secure message bodies like SDP rather than message headers. Some informational header fields, such as the Subject or Organization could perhaps warrant end-to-end security. Headers defined by future SIP applications might also require obfuscation.

Another possible application of encrypting header fields is selective anonymity. A request could be constructed with a From header field that contains no personal information (for example, sip:anonymous@anonymizer.invalid). However, a second From header field containing the genuine address-of-record of the originator could be encrypted within a "message/sip" MIME body where it will only be visible to the endpoints of a dialog.

> Note that if this mechanism is used for anonymity, the From header field will no longer be usable by the recipient of a message as an index to their certificate keychain for retrieving the proper S/MIME key to associated with the sender. The message must first be decrypted, and the "inner" From header field MUST be used as an index.

In order to provide end-to-end integrity, encrypted "message/sip" MIME bodies SHOULD be signed by the sender. This creates a "multipart/signed" MIME body that contains an encrypted body and a signature, both of type "application/pkcs7-mime".

In the following example, of an encrypted and signed message, the text boxed in asterisks ("*") is encrypted:

```
     INVITE sip:bob@biloxi.com SIP/2.0
     Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
     To: Bob <sip:bob@biloxi.com>
     From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774
     Call-ID: a84b4c76e66710
     CSeq: 314159 INVITE
     Max-Forwards: 70
     Date: Thu, 21 Feb 2002 13:02:03 GMT
     Contact: <sip:pc33.atlanta.com>
     Content-Type: multipart/signed;
       protocol="application/pkcs7-signature";
       micalg=sha1; boundary=boundary42
     Content-Length: 568

     --boundary42
     Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
          name=smime.p7m
     Content-Transfer-Encoding: base64
     Content-Disposition: attachment; filename=smime.p7m
        handling=required
     Content-Length: 231

   ***********************************************************
   * Content-Type: message/sip                               *
   *                                                         *
   * INVITE sip:bob@biloxi.com SIP/2.0                       *
   * Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 *
   * To: Bob <bob@biloxi.com>                                *
   * From: Alice <alice@atlanta.com>;tag=1928301774          *
   * Call-ID: a84b4c76e66710                                 *
   * CSeq: 314159 INVITE                                     *
   * Max-Forwards: 70                                        *
   * Date: Thu, 21 Feb 2002 13:02:03 GMT                     *
   * Contact: <sip:alice@pc33.atlanta.com>                   *
   *                                                         *
   * Content-Type: application/sdp                           *
   *                                                         *
   * v=0                                                     *
   * o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com     *
   * s=Session SDP                                           *
   * t=0 0                                                   *
   * c=IN IP4 pc33.atlanta.com                               *
   * m=audio 3456 RTP/AVP 0 1 3 99                           *
   * a=rtpmap:0 PCMU/8000                                    *
   ***********************************************************
```

```
--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
   handling=required
```


```
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756
```


```
--boundary42-
```

24 Examples

In the following examples, we often omit the message body and the corresponding Content-Length and Content-Type header fields for brevity.

24.1 Registration

Bob registers on start-up. The message flow is shown in Figure 9. Note that the authentication usually required for registration is not shown for simplicity.

Figure 9: SIP Registration Example

F1 REGISTER Bob -> Registrar

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

The registration expires after two hours. The registrar responds with a 200 OK:

F2 200 OK Registrar -> Bob

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
 ;received=192.0.2.4
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

24.2 Session Setup

This example contains the full details of the example session setup in Section 4. The message flow is shown in Figure 1. Note that these flows show the minimum required set of header fields - some other header fields such as Allow and Supported would normally be present.

F1 INVITE Alice -> atlanta.com proxy

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(Alice’s SDP not shown)

F2 100 Trying atlanta.com proxy -> Alice

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0
```

F3 INVITE atlanta.com proxy -> biloxi.com proxy

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Max-Forwards: 69
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0
```

F5 INVITE biloxi.com proxy -> Bob

```
INVITE sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
Max-Forwards: 68
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)
```

F6 180 Ringing Bob -> biloxi.com proxy

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F8 180 Ringing atlanta.com proxy -> Alice

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0
```

F9 200 OK Bob -> biloxi.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
 ;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

(Bob’s SDP not shown)

F10 200 OK biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
 ;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

(Bob’s SDP not shown)

F11 200 OK atlanta.com proxy -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
 ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)
```

F12 ACK Alice -> Bob

```
ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 ACK
Content-Length: 0
```

The media session between Alice and Bob is now established.

Bob hangs up first. Note that Bob's SIP phone maintains its own CSeq numbering space, which, in this example, begins with 231. Since Bob is making the request, the To and From URIs and tags have been swapped.

F13 BYE Bob -> Alice

```
BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
Max-Forwards: 70
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

F14 200 OK Alice -> Bob

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

The SIP Call Flows document [40] contains further examples of SIP messages.

25 Augmented BNF for the SIP Protocol

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) defined in RFC 2234 [10]. Section 6.1 of RFC 2234 defines a set of core rules that are used by this specification, and not repeated here. Implementers need to be familiar with the notation and content of RFC 2234 in order to understand this specification. Certain basic rules are in uppercase, such as SP, LWS, HTAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions to clarify the use of rule names.

The use of square brackets is redundant syntactically. It is used as a semantic hint that the specific parameter is optional to use.

25.1 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986.

```
alphanum  =  ALPHA / DIGIT
```

Several rules are incorporated from RFC 2396 [5] but are updated to make them compliant with RFC 2234 [10]. These include:

```
reserved    =  ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
               / "$" / ","
unreserved  =  alphanum / mark
mark        =  "-" / "_" / "." / "!" / "~" / "*" / "'"
               / "(" / ")"
escaped     =  "%" HEXDIG HEXDIG
```

SIP header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream. This is intended to behave exactly as HTTP/1.1 as described in RFC 2616 [8]. The SWS construct is used when linear white space is optional, generally between tokens and separators.

```
LWS  =  [*WSP CRLF] 1*WSP ; linear whitespace
SWS  =  [LWS] ; sep whitespace
```

To separate the header name from the rest of value, a colon is used, which, by the above rule, allows whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines this construct.

```
HCOLON  =  *( SP / HTAB ) ":" SWS
```

The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 charset (RFC 2279 [7]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are n t quoted strings, where leading and trailing LWS is not meaningful. In this regard, SIP differs from HTTP, which uses the ISO 8859-1 character set.

```
TEXT-UTF8-TRIM  =  1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char   =  %x21-7E / UTF8-NONASCII
UTF8-NONASCII   =  %xC0-DF 1UTF8-CONT
                /  %xE0-EF 2UTF8-CONT
                /  %xF0-F7 3UTF8-CONT
                /  %xF8-Fb 4UTF8-CONT
                /  %xFC-FD 5UTF8-CONT
UTF8-CONT       =  %x80-BF
```

A CRLF is allowed in the definition of TEXT-UTF8-TRIM only as part of a header field continuation. It is expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-UTF8-TRIM value.

Hexadecimal numeric characters are used in several protocol elements. Some elements (authentication) force hex alphas to be lower case.

```
LHEX  =  DIGIT / %x61-66 ;lowercase a-f
```

Many SIP header field values consist of words separated by LWS or special characters. Unless otherwise stated, tokens are case-insensitive. These special characters MUST be in a quoted string to be used within a parameter value. The word construct is used in Call-ID to allow most separators to be used.

```
token       =  1*(alphanum / "-" / "." / "!" / "%" / "*"
               / "_" / "+" / "`" / "'" / "~" )
separators  =  "(" / ")" / "<" / ">" / "@" /
               "," / ";" / ":" / "\" / DQUOTE /
               "/" / "[" / "]" / "?" / "=" /
               "{" / "}" / SP / HTAB
word        =  1*(alphanum / "-" / "." / "!" / "%" / "*" /
               "_" / "+" / "`" / "'" / "~" /
               "(" / ")" / "<" / ">" /
               ":" / "\" / DQUOTE /
               "/" / "[" / "]" / "?" /
               "{" / "}" )
```

When tokens are used or separators are used between elements, whitespace is often allowed before or after these characters:

```
STAR    =  SWS "*" SWS ; asterisk
SLASH   =  SWS "/" SWS ; slash
EQUAL   =  SWS "=" SWS ; equal
LPAREN  =  SWS "(" SWS ; left parenthesis
RPAREN  =  SWS ")" SWS ; right parenthesis
RAQUOT  =  ">" SWS ; right angle quote
LAQUOT  =  SWS "<"; left angle quote
COMMA   =  SWS "," SWS ; comma
SEMI    =  SWS ";" SWS ; semicolon
COLON   =  SWS ":" SWS ; colon
LDQUOT  =  SWS DQUOTE; open double quotation mark
RDQUOT  =  DQUOTE SWS ; close double quotation mark
```

Comments can be included in some SIP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition. In all other fields, parentheses are considered part of the field value.

```
   comment  =  LPAREN *(ctext / quoted-pair / comment) RPAREN
   ctext    =  %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
               / LWS
```

ctext includes all chars except left and right parens and backslash. A string of text is parsed as a single word if it is quoted using double-quote marks. In quoted strings, quotation marks (") and backslashes (\) need to be escaped.

```
   quoted-string  =  SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
   qdtext         =  LWS / %x21 / %x23-5B / %x5D-7E
                     / UTF8-NONASCII
```

The backslash character ("\") MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this mechanism to avoid conflict with line folding and header separation.

```
quoted-pair  =  "\" (%x00-09 / %x0B-0C
                / %x0E-7F)

SIP-URI          =  "sip:" [ userinfo ] hostport
                    uri-parameters [ headers ]
SIPS-URI         =  "sips:" [ userinfo ] hostport
                    uri-parameters [ headers ]
userinfo         =  ( user / telephone-subscriber ) [ ":" password ] "@"
user             =  1*( unreserved / escaped / user-unreserved )
user-unreserved  =  "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password         =  *( unreserved / escaped /
                    "&" / "=" / "+" / "$" / "," )
hostport         =  host [ ":" port ]
host             =  hostname / IPv4address / IPv6reference
hostname         =  *( domainlabel "." ) toplabel [ "." ]
domainlabel      =  alphanum
                    / alphanum *( alphanum / "-" ) alphanum
toplabel         =  ALPHA / ALPHA *( alphanum / "-" ) alphanum
```

```
IPv4address    =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference  =  "[" IPv6address "]"
IPv6address    =  hexpart [ ":" IPv4address ]
hexpart        =  hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq         =  hex4 *( ":" hex4)
hex4           =  1*4HEXDIG
port           =  1*DIGIT
```

The BNF for telephone-subscriber can be found in RFC 2806 [9]. Note, however, that any characters allowed there that are not allowed in the user part of the SIP URI MUST be escaped.

```
uri-parameters    =  *( ";" uri-parameter)
uri-parameter     =  transport-param / user-param / method-param
                     / ttl-param / maddr-param / lr-param / other-param
transport-param   =  "transport="
                     ( "udp" / "tcp" / "sctp" / "tls"
                     / other-transport)
other-transport   =  token
user-param        =  "user=" ( "phone" / "ip" / other-user)
other-user        =  token
method-param      =  "method=" Method
ttl-param         =  "ttl=" ttl
maddr-param       =  "maddr=" host
lr-param          =  "lr"
other-param       =  pname [ "=" pvalue ]
pname             =  1*paramchar
pvalue            =  1*paramchar
paramchar         =  param-unreserved / unreserved / escaped
param-unreserved  =  "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers         =  "?" header *( "&" header )
header          =  hname "=" hvalue
hname           =  1*( hnv-unreserved / unreserved / escaped )
hvalue          =  *( hnv-unreserved / unreserved / escaped )
hnv-unreserved  =  "[" / "]" / "/" / "?" / ":" / "+" / "$"

SIP-message    =  Request / Response
Request        =  Request-Line
                  *( message-header )
                  CRLF
                  [ message-body ]
Request-Line   =  Method SP Request-URI SP SIP-Version CRLF
Request-URI    =  SIP-URI / SIPS-URI / absoluteURI
absoluteURI    =  scheme ":" ( hier-part / opaque-part )
hier-part      =  ( net-path / abs-path ) [ "?" query ]
net-path       =  "//" authority [ abs-path ]
abs-path       =  "/" path-segments
```

```
opaque-part    =  uric-no-slash *uric
uric           =  reserved / unreserved / escaped
uric-no-slash  =  unreserved / escaped / ";" / "?" / ":" / "@"
                  / "&" / "=" / "+" / "$" / ","
path-segments  =  segment *( "/" segment )
segment        =  *pchar *( ";" param )
param          =  *pchar
pchar          =  unreserved / escaped /
                  ":" / "@" / "&" / "=" / "+" / "$" / ","
scheme         =  ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
authority      =  srvr / reg-name
srvr           =  [ [ userinfo "@" ] hostport ]
reg-name       =  1*( unreserved / escaped / "$" / ","
                  / ";" / ":" / "@" / "&" / "=" / "+" )
query          =  *uric
SIP-Version    =  "SIP" "/" 1*DIGIT "." 1*DIGIT

message-header  =  (Accept
                /  Accept-Encoding
                /  Accept-Language
                /  Alert-Info
                /  Allow
                /  Authentication-Info
                /  Authorization
                /  Call-ID
                /  Call-Info
                /  Contact
                /  Content-Disposition
                /  Content-Encoding
                /  Content-Language
                /  Content-Length
                /  Content-Type
                /  CSeq
                /  Date
                /  Error-Info
                /  Expires
                /  From
                /  In-Reply-To
                /  Max-Forwards
                /  MIME-Version
                /  Min-Expires
                /  Organization
                /  Priority
                /  Proxy-Authenticate
                /  Proxy-Authorization
                /  Proxy-Require
                /  Record-Route
                /  Reply-To
```

```
                /  Require
                /  Retry-After
                /  Route
                /  Server
                /  Subject
                /  Supported
                /  Timestamp
                /  To
                /  Unsupported
                /  User-Agent
                /  Via
                /  Warning
                /  WWW-Authenticate
                /  extension-header) CRLF

INVITEm           =  %x49.4E.56.49.54.45 ; INVITE in caps
ACKm              =  %x41.43.4B ; ACK in caps
OPTIONSm          =  %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm              =  %x42.59.45 ; BYE in caps
CANCELm           =  %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm         =  %x52.45.47.49.53.54.45.52 ; REGISTER in caps
Method            =  INVITEm / ACKm / OPTIONSm / BYEm
                     / CANCELm / REGISTERm
                     / extension-method
extension-method  =  token
Response          =  Status-Line
                     *( message-header )
                     CRLF
                     [ message-body ]

Status-Line     =  SIP-Version SP Status-Code SP Reason-Phrase CRLF
Status-Code     =  Informational
               /   Redirection
               /   Success
               /   Client-Error
               /   Server-Error
               /   Global-Failure
               /   extension-code
extension-code  =  3DIGIT
Reason-Phrase   =  *(reserved / unreserved / escaped
                   / UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational  =  "100"  ;  Trying
              /   "180"  ;  Ringing
              /   "181"  ;  Call Is Being Forwarded
              /   "182"  ;  Queued
              /   "183"  ;  Session Progress
```

```
Success  =  "200"  ;  OK

Redirection  =  "300"  ;  Multiple Choices
            /   "301"  ;  Moved Permanently
            /   "302"  ;  Moved Temporarily
            /   "305"  ;  Use Proxy
            /   "380"  ;  Alternative Service

Client-Error  =  "400"  ;  Bad Request
             /   "401"  ;  Unauthorized
             /   "402"  ;  Payment Required
             /   "403"  ;  Forbidden
             /   "404"  ;  Not Found
             /   "405"  ;  Method Not Allowed
             /   "406"  ;  Not Acceptable
             /   "407"  ;  Proxy Authentication Required
             /   "408"  ;  Request Timeout
             /   "410"  ;  Gone
             /   "413"  ;  Request Entity Too Large
             /   "414"  ;  Request-URI Too Large
             /   "415"  ;  Unsupported Media Type
             /   "416"  ;  Unsupported URI Scheme
             /   "420"  ;  Bad Extension
             /   "421"  ;  Extension Required
             /   "423"  ;  Interval Too Brief
             /   "480"  ;  Temporarily not available
             /   "481"  ;  Call Leg/Transaction Does Not Exist
             /   "482"  ;  Loop Detected
             /   "483"  ;  Too Many Hops
             /   "484"  ;  Address Incomplete
             /   "485"  ;  Ambiguous
             /   "486"  ;  Busy Here
             /   "487"  ;  Request Terminated
             /   "488"  ;  Not Acceptable Here
             /   "491"  ;  Request Pending
             /   "493"  ;  Undecipherable

Server-Error  =  "500"  ;  Internal Server Error
             /   "501"  ;  Not Implemented
             /   "502"  ;  Bad Gateway
             /   "503"  ;  Service Unavailable
             /   "504"  ;  Server Time-out
             /   "505"  ;  SIP Version not supported
             /   "513"  ;  Message Too Large
```

```
Global-Failure  =  "600"  ;  Busy Everywhere
               /   "603"  ;  Decline
               /   "604"  ;  Does not exist anywhere
               /   "606"  ;  Not Acceptable

Accept         =  "Accept" HCOLON
                   [ accept-range *(COMMA accept-range) ]
accept-range   =  media-range *(SEMI accept-param)
media-range    =  ( "*/*"
                  / ( m-type SLASH "*" )
                  / ( m-type SLASH m-subtype )
                  ) *( SEMI m-parameter )
accept-param   =  ("q" EQUAL qvalue) / generic-param
qvalue         =  ( "0" [ "." 0*3DIGIT ] )
                  / ( "1" [ "." 0*3("0") ] )
generic-param  =  token [ EQUAL gen-value ]
gen-value      =  token / host / quoted-string

Accept-Encoding  =  "Accept-Encoding" HCOLON
                     [ encoding *(COMMA encoding) ]
encoding         =  codings *(SEMI accept-param)
codings          =  content-coding / "*"
content-coding   =  token

Accept-Language  =  "Accept-Language" HCOLON
                     [ language *(COMMA language) ]
language         =  language-range *(SEMI accept-param)
language-range   =  ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )

Alert-Info   =  "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param  =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Allow  =  "Allow" HCOLON [Method *(COMMA Method)]

Authorization     =  "Authorization" HCOLON credentials
credentials       =  ("Digest" LWS digest-response)
                     / other-response
digest-response   =  dig-resp *(COMMA dig-resp)
dig-resp          =  username / realm / nonce / digest-uri
                      / dresponse / algorithm / cnonce
                      / opaque / message-qop
                      / nonce-count / auth-param
username          =  "username" EQUAL username-value
username-value    =  quoted-string
digest-uri        =  "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value  =  rquest-uri ; Equal to request-uri as specified
                     by HTTP/1.1
message-qop       =  "qop" EQUAL qop-value
```

```
cnonce            =  "cnonce" EQUAL cnonce-value
cnonce-value      =  nonce-value
nonce-count       =  "nc" EQUAL nc-value
nc-value          =  8LHEX
dresponse         =  "response" EQUAL request-digest
request-digest    =  LDQUOT 32LHEX RDQUOT
auth-param        =  auth-param-name EQUAL
                     ( token / quoted-string )
auth-param-name   =  token
other-response    =  auth-scheme LWS auth-param
                     *(COMMA auth-param)
auth-scheme       =  token

Authentication-Info  =  "Authentication-Info" HCOLON ainfo
                        *(COMMA ainfo)
ainfo                =  nextnonce / message-qop
                         / response-auth / cnonce
                         / nonce-count
nextnonce            =  "nextnonce" EQUAL nonce-value
response-auth        =  "rspauth" EQUAL response-digest
response-digest      =  LDQUOT *LHEX RDQUOT

Call-ID  =  ( "Call-ID" / "i" ) HCOLON callid
callid   =  word [ "@" word ]

Call-Info   =  "Call-Info" HCOLON info *(COMMA info)
info        =  LAQUOT absoluteURI RAQUOT *( SEMI info-param)
info-param  =  ( "purpose" EQUAL ( "icon" / "info"
               / "card" / token ) ) / generic-param

Contact        =  ("Contact" / "m" ) HCOLON
                  ( STAR / (contact-param *(COMMA contact-param)))
contact-param  =  (name-addr / addr-spec) *(SEMI contact-params)
name-addr      =  [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec      =  SIP-URI / SIPS-URI / absoluteURI
display-name   =  *(token LWS)/ quoted-string

contact-params     =  c-p-q / c-p-expires
                      / contact-extension
c-p-q              =  "q" EQUAL qvalue
c-p-expires        =  "expires" EQUAL delta-seconds
contact-extension  =  generic-param
delta-seconds      =  1*DIGIT

Content-Disposition   =  "Content-Disposition" HCOLON
                         disp-type *( SEMI disp-param )
disp-type             =  "render" / "session" / "icon" / "alert"
                         / disp-extension-token
```

```
disp-param             =  handling-param / generic-param
handling-param         =  "handling" EQUAL
                          ( "optional" / "required"
                          / other-handling )
other-handling         =  token
disp-extension-token   =  token

Content-Encoding  =  ( "Content-Encoding" / "e" ) HCOLON
                     content-coding *(COMMA content-coding)

Content-Language  =  "Content-Language" HCOLON
                     language-tag *(COMMA language-tag)
language-tag      =  primary-tag *( "-" subtag )
primary-tag       =  1*8ALPHA
subtag            =  1*8ALPHA

Content-Length  =  ( "Content-Length" / "l" ) HCOLON 1*DIGIT
Content-Type     =  ( "Content-Type" / "c" ) HCOLON media-type
media-type       =  m-type SLASH m-subtype *(SEMI m-parameter)
m-type           =  discrete-type / composite-type
discrete-type    =  "text" / "image" / "audio" / "video"
                    / "application" / extension-token
composite-type   =  "message" / "multipart" / extension-token
extension-token  =  ietf-token / x-token
ietf-token       =  token
x-token          =  "x-" token
m-subtype        =  extension-token / iana-token
iana-token       =  token
m-parameter      =  m-attribute EQUAL m-value
m-attribute      =  token
m-value          =  token / quoted-string

CSeq  =  "CSeq" HCOLON 1*DIGIT LWS Method

Date          =  "Date" HCOLON SIP-date
SIP-date      =  rfc1123-date
rfc1123-date  =  wkday "," SP date1 SP time SP "GMT"
date1         =  2DIGIT SP month SP 4DIGIT
                 ; day month year (e.g., 02 Jun 1982)
time          =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                 ; 00:00:00 - 23:59:59
wkday         =  "Mon" / "Tue" / "Wed"
                 / "Thu" / "Fri" / "Sat" / "Sun"
month         =  "Jan" / "Feb" / "Mar" / "Apr"
                 / "May" / "Jun" / "Jul" / "Aug"
                 / "Sep" / "Oct" / "Nov" / "Dec"

Error-Info  =  "Error-Info" HCOLON error-uri *(COMMA error-uri)
```

```
error-uri   =  LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Expires     =  "Expires" HCOLON delta-seconds
From        =  ( "From" / "f" ) HCOLON from-spec
from-spec   =  ( name-addr / addr-spec )
               *( SEMI from-param )
from-param  =  tag-param / generic-param
tag-param   =  "tag" EQUAL token

In-Reply-To  =  "In-Reply-To" HCOLON callid *(COMMA callid)

Max-Forwards  =  "Max-Forwards" HCOLON 1*DIGIT

MIME-Version  =  "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

Min-Expires  =  "Min-Expires" HCOLON delta-seconds

Organization  =  "Organization" HCOLON [TEXT-UTF8-TRIM]

Priority        =  "Priority" HCOLON priority-value
priority-value  =  "emergency" / "urgent" / "normal"
                   / "non-urgent" / other-priority
other-priority  =  token

Proxy-Authenticate  =  "Proxy-Authenticate" HCOLON challenge
challenge           =  ("Digest" LWS digest-cln *(COMMA digest-cln))
                       / other-challenge
other-challenge     =  auth-scheme LWS auth-param
                       *(COMMA auth-param)
digest-cln          =  realm / domain / nonce
                        / opaque / stale / algorithm
                        / qop-options / auth-param
realm               =  "realm" EQUAL realm-value
realm-value         =  quoted-string
domain              =  "domain" EQUAL LDQUOT URI
                       *( 1*SP URI ) RDQUOT
URI                 =  absoluteURI / abs-path
nonce               =  "nonce" EQUAL nonce-value
nonce-value         =  quoted-string
opaque              =  "opaque" EQUAL quoted-string
stale               =  "stale" EQUAL ( "true" / "false" )
algorithm           =  "algorithm" EQUAL ( "MD5" / "MD5-sess"
                       / token )
qop-options         =  "qop" EQUAL LDQUOT qop-value
                       *("," qop-value) RDQUOT
qop-value           =  "auth" / "auth-int" / token

Proxy-Authorization  =  "Proxy-Authorization" HCOLON credentials
```

```
Proxy-Require  =  "Proxy-Require" HCOLON option-tag
                  *(COMMA option-tag)
option-tag     =  token

Record-Route  =  "Record-Route" HCOLON rec-route *(COMMA rec-route)
rec-route     =  name-addr *( SEMI rr-param )
rr-param      =  generic-param

Reply-To      =  "Reply-To" HCOLON rplyto-spec
rplyto-spec   =  ( name-addr / addr-spec )
                 *( SEMI rplyto-param )
rplyto-param  =  generic-param
Require       =  "Require" HCOLON option-tag *(COMMA option-tag)

Retry-After  =  "Retry-After" HCOLON delta-seconds
                [ comment ] *( SEMI retry-param )

retry-param  =  ("duration" EQUAL delta-seconds)
                / generic-param

Route        =  "Route" HCOLON route-param *(COMMA route-param)
route-param  =  name-addr *( SEMI rr-param )

Server           =  "Server" HCOLON server-val *(LWS server-val)
server-val       =  product / comment
product          =  token [SLASH product-version]
product-version  =  token

Subject  =  ( "Subject" / "s" ) HCOLON [TEXT-UTF8-TRIM]

Supported  =  ( "Supported" / "k" ) HCOLON
              [option-tag *(COMMA option-tag)]

Timestamp  =  "Timestamp" HCOLON 1*(DIGIT)
               [ "." *(DIGIT) ] [ LWS delay ]
delay      =  *(DIGIT) [ "." *(DIGIT) ]

To        =  ( "To" / "t" ) HCOLON ( name-addr
             / addr-spec ) *( SEMI to-param )
to-param  =  tag-param / generic-param

Unsupported  =  "Unsupported" HCOLON option-tag *(COMMA option-tag)
User-Agent  =  "User-Agent" HCOLON server-val *(LWS server-val)
```

```
Via               =  ( "Via" / "v" ) HCOLON via-parm *(COMMA via-parm)
via-parm          =  sent-protocol LWS sent-by *( SEMI via-params )
via-params        =  via-ttl / via-maddr
                     / via-received / via-branch
                     / via-extension
via-ttl           =  "ttl" EQUAL ttl
via-maddr         =  "maddr" EQUAL host
via-received      =  "received" EQUAL (IPv4address / IPv6address)
via-branch        =  "branch" EQUAL token
via-extension     =  generic-param
sent-protocol     =  protocol-name SLASH protocol-version
                     SLASH transport
protocol-name     =  "SIP" / token
protocol-version  =  token
transport         =  "UDP" / "TCP" / "TLS" / "SCTP"
                     / other-transport
sent-by           =  host [ COLON port ]
ttl               =  1*3DIGIT ; 0 to 255

Warning        =  "Warning" HCOLON warning-value *(COMMA warning-value)
warning-value  =  warn-code SP warn-agent SP warn-text
warn-code      =  3DIGIT
warn-agent     =  hostport / pseudonym
                  ;  the name or pseudonym of the server adding
                  ;  the Warning header, for use in debugging
warn-text      =  quoted-string
pseudonym      =  token

WWW-Authenticate  =  "WWW-Authenticate" HCOLON challenge

extension-header  =  header-name HCOLON header-value
header-name       =  token
header-value      =  *(TEXT-UTF8char / UTF8-CONT / LWS)
message-body  =  *OCTET
```

26 Security Considerations: Threat Model and Security Usage Recommendations

SIP is not an easy protocol to secure. Its use of intermediaries, its multi-faceted trust relationships, its expected usage between elements with no trust at all, and its user-to-user operation make security far from trivial. Security solutions are needed that are deployable today, without extensive coordination, in a wide variety of environments and usages. In order to meet these diverse needs, several distinct mechanisms applicable to different aspects and usages of SIP will be required.

Note that the security of SIP signaling itself has no bearing on the security of protocols used in concert with SIP such as RTP, or with the security implications of any specific bodies SIP might carry (although MIME security plays a substantial role in securing SIP). Any media associated with a session can be encrypted end-to-end independently of any associated SIP signaling. Media encryption is outside the scope of this document.

The considerations that follow first examine a set of classic threat models that broadly identify the security needs of SIP. The set of security services required to address these threats is then detailed, followed by an explanation of several security mechanisms that can be used to provide these services. Next, the requirements for implementers of SIP are enumerated, along with exemplary deployments in which these security mechanisms could be used to improve the security of SIP. Some notes on privacy conclude this section.

26.1 Attacks and Threat Models

This section details some threats that should be common to most deployments of SIP. These threats have been chosen specifically to illustrate each of the security services that SIP requires.

The following examples by no means provide an exhaustive list of the threats against SIP; rather, these are "classic" threats that demonstrate the need for particular security services that can potentially prevent whole categories of threats.

These attacks assume an environment in which attackers can potentially read any packet on the network - it is anticipated that SIP will frequently be used on the public Internet. Attackers on the network may be able to modify packets (perhaps at some compromised intermediary). Attackers may wish to steal services, eavesdrop on communications, or disrupt sessions.

26.1.1 Registration Hijacking

The SIP registration mechanism allows a user agent to identify itself to a registrar as a device at which a user (designated by an address of record) is located. A registrar assesses the identity asserted in the From header field of a REGISTER message to determine whether this request can modify the contact addresses associated with the address-of-record in the To header field. While these two fields are frequently the same, there are many valid deployments in which a third-party may register contacts on a user's behalf.

The From header field of a SIP request, however, can be modified arbitrarily by the owner of a UA, and this opens the door to malicious registrations. An attacker that successfully impersonates a party authorized to change contacts associated with an address-of-record could, for example, de-register all existing contacts for a URI and then register their own device as the appropriate contact address, thereby directing all requests for the affected user to the attacker's device.

This threat belongs to a family of threats that rely on the absence of cryptographic assurance of a request's originator. Any SIP UAS that represents a valuable service (a gateway that interworks SIP requests with traditional telephone calls, for example) might want to control access to its resources by authenticating requests that it receives. Even end-user UAs, for example SIP phones, have an interest in ascertaining the identities of originators of requests.

This threat demonstrates the need for security services that enable SIP entities to authenticate the originators of requests.

26.1.2 Impersonating a Server

The domain to which a request is destined is generally specified in the Request-URI. UAs commonly contact a server in this domain directly in order to deliver a request. However, there is always a possibility that an attacker could impersonate the remote server, and that the UA's request could be intercepted by some other party.

For example, consider a case in which a redirect server at one domain, chicago.com, impersonates a redirect server at another domain, biloxi.com. A user agent sends a request to biloxi.com, but the redirect server at chicago.com answers with a forged response that has appropriate SIP header fields for a response from biloxi.com. The forged contact addresses in the redirection response could direct the originating UA to inappropriate or insecure resources, or simply prevent requests for biloxi.com from succeeding.

This family of threats has a vast membership, many of which are critical. As a converse to the registration hijacking threat, consider the case in which a registration sent to biloxi.com is intercepted by chicago.com, which replies to the intercepted registration with a forged 301 (Moved Permanently) response. This response might seem to come from biloxi.com yet designate chicago.com as the appropriate registrar. All future REGISTER requests from the originating UA would then go to chicago.com.

Prevention of this threat requires a means by which UAs can authenticate the servers to whom they send requests.

26.1.3 Tampering with Message Bodies

As a matter of course, SIP UAs route requests through trusted proxy servers. Regardless of how that trust is established (authentication of proxies is discussed elsewhere in this section), a UA may trust a proxy server to route a request, but not to inspect or possibly modify the bodies contained in that request.

Consider a UA that is using SIP message bodies to communicate session encryption keys for a media session. Although it trusts the proxy server of the domain it is contacting to deliver signaling properly, it may not want the administrators of that domain to be capable of decrypting any subsequent media session. Worse yet, if the proxy server were actively malicious, it could modify the session key, either acting as a man-in-the-middle, or perhaps changing the security characteristics requested by the originating UA.

This family of threats applies not only to session keys, but to most conceivable forms of content carried end-to-end in SIP. These might include MIME bodies that should be rendered to the user, SDP, or encapsulated telephony signals, among others. Attackers might attempt to modify SDP bodies, for example, in order to point RTP media streams to a wiretapping device in order to eavesdrop on subsequent voice communications.

Also note that some header fields in SIP are meaningful end-to-end, for example, Subject. UAs might be protective of these header fields as well as bodies (a malicious intermediary changing the Subject header field might make an important request appear to be spam, for example). However, since many header fields are legitimately inspected or altered by proxy servers as a request is routed, not all header fields should be secured end-to-end.

For these reasons, the UA might want to secure SIP message bodies, and in some limited cases header fields, end-to-end. The security services required for bodies include confidentiality, integrity, and authentication. These end-to-end services should be independent of the means used to secure interactions with intermediaries such as proxy servers.

26.1.4 Tearing Down Sessions

Once a dialog has been established by initial messaging, subsequent requests can be sent that modify the state of the dialog and/or session. It is critical that principals in a session can be certain that such requests are not forged by attackers.

Consider a case in which a third-party attacker captures some initial messages in a dialog shared by two parties in order to learn the parameters of the session (To tag, From tag, and so forth) and then inserts a BYE request into the session. The attacker could opt to forge the request such that it seemed to come from either participant. Once the BYE is received by its target, the session will be torn down prematurely.

Similar mid-session threats include the transmission of forged re-INVITEs that alter the session (possibly to reduce session security or redirect media streams as part of a wiretapping attack).

The most effective countermeasure to this threat is the authentication of the sender of the BYE. In this instance, the recipient needs only know that the BYE came from the same party with whom the corresponding dialog was established (as opposed to ascertaining the absolute identity of the sender). Also, if the attacker is unable to learn the parameters of the session due to confidentiality, it would not be possible to forge the BYE. However, some intermediaries (like proxy servers) will need to inspect those parameters as the session is established.

26.1.5 Denial of Service and Amplification

Denial-of-service attacks focus on rendering a particular network element unavailable, usually by directing an excessive amount of network traffic at its interfaces. A distributed denial-of-service attack allows one network user to cause multiple network hosts to flood a target host with a large amount of network traffic.

In many architectures, SIP proxy servers face the public Internet in order to accept requests from worldwide IP endpoints. SIP creates a number of potential opportunities for distributed denial-of-service attacks that must be recognized and addressed by the implementers and operators of SIP systems.

Attackers can create bogus requests that contain a falsified source IP address and a corresponding Via header field that identify a targeted host as the originator of the request and then send this request to a large number of SIP network elements, thereby using hapless SIP UAs or proxies to generate denial-of-service traffic aimed at the target.

Similarly, attackers might use falsified Route header field values in a request that identify the target host and then send such messages to forking proxies that will amplify messaging sent to the target.

Record-Route could be used to similar effect when the attacker is certain that the SIP dialog initiated by the request will result in numerous transactions originating in the backwards direction.

A number of denial-of-service attacks open up if REGISTER requests are not properly authenticated and authorized by registrars. Attackers could de-register some or all users in an administrative domain, thereby preventing these users from being invited to new sessions. An attacker could also register a large number of contacts designating the same host for a given address-of-record in order to use the registrar and any associated proxy servers as amplifiers in a denial-of-service attack. Attackers might also attempt to deplete available memory and disk resources of a registrar by registering huge numbers of bindings.

The use of multicast to transmit SIP requests can greatly increase the potential for denial-of-service attacks.

These problems demonstrate a general need to define architectures that minimize the risks of denial-of-service, and the need to be mindful in recommendations for security mechanisms of this class of attacks.

26.2 Security Mechanisms

From the threats described above, we gather that the fundamental security services required for the SIP protocol are: preserving the confidentiality and integrity of messaging, preventing replay attacks or message spoofing, providing for the authentication and privacy of the participants in a session, and preventing denial-of-service attacks. Bodies within SIP messages separately require the security services of confidentiality, integrity, and authentication.

Rather than defining new security mechanisms specific to SIP, SIP reuses wherever possible existing security models derived from the HTTP and SMTP space.

Full encryption of messages provides the best means to preserve the confidentiality of signaling - it can also guarantee that messages are not modified by any malicious intermediaries. However, SIP requests and responses cannot be naively encrypted end-to-end in their entirety because message fields such as the Request-URI, Route, and Via need to be visible to proxies in most network architectures so that SIP requests are routed correctly. Note that proxy servers need to modify some features of messages as well (such as adding Via header field values) in order for SIP to function. Proxy servers must therefore be trusted, to some degree, by SIP UAs. To this purpose, low-layer security mechanisms for SIP are recommended, which

encrypt the entire SIP requests or responses on the wire on a hop-by-hop basis, and that allow endpoints to verify the identity of proxy servers to whom they send requests.

SIP entities also have a need to identify one another in a secure fashion. When a SIP endpoint asserts the identity of its user to a peer UA or to a proxy server, that identity should in some way be verifiable. A cryptographic authentication mechanism is provided in SIP to address this requirement.

An independent security mechanism for SIP message bodies supplies an alternative means of end-to-end mutual authentication, as well as providing a limit on the degree to which user agents must trust intermediaries.

26.2.1 Transport and Network Layer Security

Transport or network layer security encrypts signaling traffic, guaranteeing message confidentiality and integrity.

Oftentimes, certificates are used in the establishment of lower-layer security, and these certificates can also be used to provide a means of authentication in many architectures.

Two popular alternatives for providing security at the transport and network layer are, respectively, TLS [25] and IPSec [26].

IPSec is a set of network-layer protocol tools that collectively can be used as a secure replacement for traditional IP (Internet Protocol). IPSec is most commonly used in architectures in which a set of hosts or administrative domains have an existing trust relationship with one another. IPSec is usually implemented at the operating system level in a host, or on a security gateway that provides confidentiality and integrity for all traffic it receives from a particular interface (as in a VPN architecture). IPSec can also be used on a hop-by-hop basis.

In many architectures IPSec does not require integration with SIP applications; IPSec is perhaps best suited to deployments in which adding security directly to SIP hosts would be arduous. UAs that have a pre-shared keying relationship with their first-hop proxy server are also good candidates to use IPSec. Any deployment of IPSec for SIP would require an IPSec profile describing the protocol tools that would be required to secure SIP. No such profile is given in this document.

TLS provides transport-layer security over connection-oriented protocols (for the purposes of this document, TCP); "tls" (signifying TLS over TCP) can be specified as the desired transport protocol within a Via header field value or a SIP-URI. TLS is most suited to architectures in which hop-by-hop security is required between hosts with no pre-existing trust association. For example, Alice trusts her local proxy server, which after a certificate exchange decides to trust Bob's local proxy server, which Bob trusts, hence Bob and Alice can communicate securely.

TLS must be tightly coupled with a SIP application. Note that transport mechanisms are specified on a hop-by-hop basis in SIP, thus a UA that sends requests over TLS to a proxy server has no assurance that TLS will be used end-to-end.

The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite [6] MUST be supported at a minimum by implementers when TLS is used in a SIP application. For purposes of backwards compatibility, proxy servers, redirect servers, and registrars SHOULD support TLS_RSA_WITH_3DES_EDE_CBC_SHA. Implementers MAY also support any other ciphersuite.

26.2.2 SIPS URI Scheme

The SIPS URI scheme adheres to the syntax of the SIP URI (described in 19), although the scheme string is "sips" rather than "sip". The semantics of SIPS are very different from the SIP URI, however. SIPS allows resources to specify that they should be reached securely.

A SIPS URI can be used as an address-of-record for a particular user - the URI by which the user is canonically known (on their business cards, in the From header field of their requests, in the To header field of REGISTER requests). When used as the Request-URI of a request, the SIPS scheme signifies that each hop over which the request is forwarded, until the request reaches the SIP entity responsible for the domain portion of the Request-URI, must be secured with TLS; once it reaches the domain in question it is handled in accordance with local security and routing policy, quite possibly using TLS for any last hop to a UAS. When used by the originator of a request (as would be the case if they employed a SIPS URI as the address-of-record of the target), SIPS dictates that the entire request path to the target domain be so secured.

The SIPS scheme is applicable to many of the other ways in which SIP URIs are used in SIP today in addition to the Request-URI, including in addresses-of-record, contact addresses (the contents of Contact headers, including those of REGISTER methods), and Route headers. In each instance, the SIPS URI scheme allows these existing fields to

designate secure resources. The manner in which a SIPS URI is dereferenced in any of these contexts has its own security properties which are detailed in [4].

The use of SIPS in particular entails that mutual TLS authentication SHOULD be employed, as SHOULD the ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA. Certificates received in the authentication process SHOULD be validated with root certificates held by the client; failure to validate a certificate SHOULD result in the failure of the request.

> Note that in the SIPS URI scheme, transport is independent of TLS, and thus "sips:alice@atlanta.com;transport=tcp" and "sips:alice@atlanta.com;transport=sctp" are both valid (although note that UDP is not a valid transport for SIPS). The use of "transport=tls" has consequently been deprecated, partly because it was specific to a single hop of the request. This is a change since RFC 2543.

Users that distribute a SIPS URI as an address-of-record may elect to operate devices that refuse requests over insecure transports.

26.2.3 HTTP Authentication

SIP provides a challenge capability, based on HTTP authentication, that relies on the 401 and 407 response codes as well as header fields for carrying challenges and credentials. Without significant modification, the reuse of the HTTP Digest authentication scheme in SIP allows for replay protection and one-way authentication.

The usage of Digest authentication in SIP is detailed in Section 22.

26.2.4 S/MIME

As is discussed above, encrypting entire SIP messages end-to-end for the purpose of confidentiality is not appropriate because network intermediaries (like proxy servers) need to view certain header fields in order to route messages correctly, and if these intermediaries are excluded from security associations, then SIP messages will essentially be non-routable.

However, S/MIME allows SIP UAs to encrypt MIME bodies within SIP, securing these bodies end-to-end without affecting message headers. S/MIME can provide end-to-end confidentiality and integrity for message bodies, as well as mutual authentication. It is also possible to use S/MIME to provide a form of integrity and confidentiality for SIP header fields through SIP message tunneling.

The usage of S/MIME in SIP is detailed in Section 23.

26.3 Implementing Security Mechanisms

26.3.1 Requirements for Implementers of SIP

Proxy servers, redirect servers, and registrars MUST implement TLS, and MUST support both mutual and one-way authentication. It is strongly RECOMMENDED that UAs be capable initiating TLS; UAs MAY also be capable of acting as a TLS server. Proxy servers, redirect servers, and registrars SHOULD possess a site certificate whose subject corresponds to their canonical hostname. UAs MAY have certificates of their own for mutual authentication with TLS, but no provisions are set forth in this document for their use. All SIP elements that support TLS MUST have a mechanism for validating certificates received during TLS negotiation; this entails possession of one or more root certificates issued by certificate authorities (preferably well-known distributors of site certificates comparable to those that issue root certificates for web browsers).

All SIP elements that support TLS MUST also support the SIPS URI scheme.

Proxy servers, redirect servers, registrars, and UAs MAY also implement IPSec or other lower-layer security protocols.

When a UA attempts to contact a proxy server, redirect server, or registrar, the UAC SHOULD initiate a TLS connection over which it will send SIP messages. In some architectures, UASs MAY receive requests over such TLS connections as well.

Proxy servers, redirect servers, registrars, and UAs MUST implement Digest Authorization, encompassing all of the aspects required in 22. Proxy servers, redirect servers, and registrars SHOULD be configured with at least one Digest realm, and at least one "realm" string supported by a given server SHOULD correspond to the server's hostname or domainname.

UAs MAY support the signing and encrypting of MIME bodies, and transference of credentials with S/MIME as described in Section 23. If a UA holds one or more root certificates of certificate authorities in order to validate certificates for TLS or IPSec, it SHOULD be capable of reusing these to verify S/MIME certificates, as appropriate. A UA MAY hold root certificates specifically for validating S/MIME certificates.

Note that is it anticipated that future security extensions may upgrade the normative strength associated with S/MIME as S/MIME implementations appear and the problem space becomes better understood.

26.3.2 Security Solutions

The operation of these security mechanisms in concert can follow the existing web and email security models to some degree. At a high level, UAs authenticate themselves to servers (proxy servers, redirect servers, and registrars) with a Digest username and password; servers authenticate themselves to UAs one hop away, or to another server one hop away (and vice versa), with a site certificate delivered by TLS.

On a peer-to-peer level, UAs trust the network to authenticate one another ordinarily; however, S/MIME can also be used to provide direct authentication when the network does not, or if the network itself is not trusted.

The following is an illustrative example in which these security mechanisms are used by various UAs and servers to prevent the sorts of threats described in Section 26.1. While implementers and network administrators MAY follow the normative guidelines given in the remainder of this section, these are provided only as example implementations.

26.3.2.1 Registration

When a UA comes online and registers with its local administrative domain, it SHOULD establish a TLS connection with its registrar (Section 10 describes how the UA reaches its registrar). The registrar SHOULD offer a certificate to the UA, and the site identified by the certificate MUST correspond with the domain in which the UA intends to register; for example, if the UA intends to register the address-of-record 'alice@atlanta.com', the site certificate must identify a host within the atlanta.com domain (such as sip.atlanta.com). When it receives the TLS Certificate message, the UA SHOULD verify the certificate and inspect the site identified by the certificate. If the certificate is invalid, revoked, or if it does not identify the appropriate party, the UA MUST NOT send the REGISTER message and otherwise proceed with the registration.

When a valid certificate has been provided by the registrar, the UA knows that the registrar is not an attacker who might redirect the UA, steal passwords, or attempt any similar attacks.

The UA then creates a REGISTER request that SHOULD be addressed to a Request-URI corresponding to the site certificate received from the registrar. When the UA sends the REGISTER request over the existing TLS connection, the registrar SHOULD challenge the request with a 401 (Proxy Authentication Required) response. The "realm" parameter within the Proxy-Authenticate header field of the response SHOULD correspond to the domain previously given by the site certificate. When the UAC receives the challenge, it SHOULD either prompt the user for credentials or take an appropriate credential from a keyring corresponding to the "realm" parameter in the challenge. The username of this credential SHOULD correspond with the "userinfo" portion of the URI in the To header field of the REGISTER request. Once the Digest credentials have been inserted into an appropriate Proxy-Authorization header field, the REGISTER should be resubmitted to the registrar.

> Since the registrar requires the user agent to authenticate itself, it would be difficult for an attacker to forge REGISTER requests for the user's address-of-record. Also note that since the REGISTER is sent over a confidential TLS connection, attackers will not be able to intercept the REGISTER to record credentials for any possible replay attack.

Once the registration has been accepted by the registrar, the UA SHOULD leave this TLS connection open provided that the registrar also acts as the proxy server to which requests are sent for users in this administrative domain. The existing TLS connection will be reused to deliver incoming requests to the UA that has just completed registration.

> Because the UA has already authenticated the server on the other side of the TLS connection, all requests that come over this connection are known to have passed through the proxy server - attackers cannot create spoofed requests that appear to have been sent through that proxy server.

26.3.2.2 Interdomain Requests

Now let's say that Alice's UA would like to initiate a session with a user in a remote administrative domain, namely "bob@biloxi.com". We will also say that the local administrative domain (atlanta.com) has a local outbound proxy.

The proxy server that handles inbound requests for an administrative domain MAY also act as a local outbound proxy; for simplicity's sake we'll assume this to be the case for atlanta.com (otherwise the user agent would initiate a new TLS connection to a separate server at this point). Assuming that the client has completed the registration

process described in the preceding section, it SHOULD reuse the TLS connection to the local proxy server when it sends an INVITE request to another user. The UA SHOULD reuse cached credentials in the INVITE to avoid prompting the user unnecessarily.

When the local outbound proxy server has validated the credentials presented by the UA in the INVITE, it SHOULD inspect the Request-URI to determine how the message should be routed (see [4]). If the "domainname" portion of the Request-URI had corresponded to the local domain (atlanta.com) rather than biloxi.com, then the proxy server would have consulted its location service to determine how best to reach the requested user.

> Had "alice@atlanta.com" been attempting to contact, say, "alex@atlanta.com", the local proxy would have proxied to the request to the TLS connection Alex had established with the registrar when he registered. Since Alex would receive this request over his authenticated channel, he would be assured that Alice's request had been authorized by the proxy server of the local administrative domain.

However, in this instance the Request-URI designates a remote domain. The local outbound proxy server at atlanta.com SHOULD therefore establish a TLS connection with the remote proxy server at biloxi.com. Since both of the participants in this TLS connection are servers that possess site certificates, mutual TLS authentication SHOULD occur. Each side of the connection SHOULD verify and inspect the certificate of the other, noting the domain name that appears in the certificate for comparison with the header fields of SIP messages. The atlanta.com proxy server, for example, SHOULD verify at this stage that the certificate received from the remote side corresponds with the biloxi.com domain. Once it has done so, and TLS negotiation has completed, resulting in a secure channel between the two proxies, the atlanta.com proxy can forward the INVITE request to biloxi.com.

The proxy server at biloxi.com SHOULD inspect the certificate of the proxy server at atlanta.com in turn and compare the domain asserted by the certificate with the "domainname" portion of the From header field in the INVITE request. The biloxi proxy MAY have a strict security policy that requires it to reject requests that do not match the administrative domain from which they have been proxied.

> Such security policies could be instituted to prevent the SIP equivalent of SMTP 'open relays' that are frequently exploited to generate spam.

This policy, however, only guarantees that the request came from the domain it ascribes to itself; it does not allow biloxi.com to ascertain how atlanta.com authenticated Alice. Only if biloxi.com has some other way of knowing atlanta.com's authentication policies could it possibly ascertain how Alice proved her identity. biloxi.com might then institute an even stricter policy that forbids requests that come from domains that are not known administratively to share a common authentication policy with biloxi.com.

Once the INVITE has been approved by the biloxi proxy, the proxy server SHOULD identify the existing TLS channel, if any, associated with the user targeted by this request (in this case "bob@biloxi.com"). The INVITE should be proxied through this channel to Bob. Since the request is received over a TLS connection that had previously been authenticated as the biloxi proxy, Bob knows that the From header field was not tampered with and that atlanta.com has validated Alice, although not necessarily whether or not to trust Alice's identity.

Before they forward the request, both proxy servers SHOULD add a Record-Route header field to the request so that all future requests in this dialog will pass through the proxy servers. The proxy servers can thereby continue to provide security services for the lifetime of this dialog. If the proxy servers do not add themselves to the Record-Route, future messages will pass directly end-to-end between Alice and Bob without any security services (unless the two parties agree on some independent end-to-end security such as S/MIME). In this respect the SIP trapezoid model can provide a nice structure where conventions of agreement between the site proxies can provide a reasonably secure channel between Alice and Bob.

> An attacker preying on this architecture would, for example, be unable to forge a BYE request and insert it into the signaling stream between Bob and Alice because the attacker has no way of ascertaining the parameters of the session and also because the integrity mechanism transitively protects the traffic between Alice and Bob.

26.3.2.3 Peer-to-Peer Requests

Alternatively, consider a UA asserting the identity "carol@chicago.com" that has no local outbound proxy. When Carol wishes to send an INVITE to "bob@biloxi.com", her UA SHOULD initiate a TLS connection with the biloxi proxy directly (using the mechanism described in [4] to determine how to best to reach the given Request-URI). When her UA receives a certificate from the biloxi proxy, it SHOULD be verified normally before she passes her INVITE across the TLS connection. However, Carol has no means of proving

her identity to the biloxi proxy, but she does have a CMS-detached signature over a "message/sip" body in the INVITE. It is unlikely in this instance that Carol would have any credentials in the biloxi.com realm, since she has no formal association with biloxi.com. The biloxi proxy MAY also have a strict policy that precludes it from even bothering to challenge requests that do not have biloxi.com in the "domainname" portion of the From header field - it treats these users as unauthenticated.

The biloxi proxy has a policy for Bob that all non-authenticated requests should be redirected to the appropriate contact address registered against 'bob@biloxi.com', namely <sip:bob@192.0.2.4>. Carol receives the redirection response over the TLS connection she established with the biloxi proxy, so she trusts the veracity of the contact address.

Carol SHOULD then establish a TCP connection with the designated address and send a new INVITE with a Request-URI containing the received contact address (recomputing the signature in the body as the request is readied). Bob receives this INVITE on an insecure interface, but his UA inspects and, in this instance, recognizes the From header field of the request and subsequently matches a locally cached certificate with the one presented in the signature of the body of the INVITE. He replies in similar fashion, authenticating himself to Carol, and a secure dialog begins.

> Sometimes firewalls or NATs in an administrative domain could preclude the establishment of a direct TCP connection to a UA. In these cases, proxy servers could also potentially relay requests to UAs in a way that has no trust implications (for example, forgoing an existing TLS connection and forwarding the request over cleartext TCP) as local policy dictates.

26.3.2.4 DoS Protection

In order to minimize the risk of a denial-of-service attack against architectures using these security solutions, implementers should take note of the following guidelines.

When the host on which a SIP proxy server is operating is routable from the public Internet, it SHOULD be deployed in an administrative domain with defensive operational policies (blocking source-routed traffic, preferably filtering ping traffic). Both TLS and IPSec can also make use of bastion hosts at the edges of administrative domains that participate in the security associations to aggregate secure tunnels and sockets. These bastion hosts can also take the brunt of denial-of-service attacks, ensuring that SIP hosts within the administrative domain are not encumbered with superfluous messaging.

No matter what security solutions are deployed, floods of messages directed at proxy servers can lock up proxy server resources and prevent desirable traffic from reaching its destination. There is a computational expense associated with processing a SIP transaction at a proxy server, and that expense is greater for stateful proxy servers than it is for stateless proxy servers. Therefore, stateful proxies are more susceptible to flooding than stateless proxy servers.

UAs and proxy servers SHOULD challenge questionable requests with only a single 401 (Unauthorized) or 407 (Proxy Authentication Required), forgoing the normal response retransmission algorithm, and thus behaving statelessly towards unauthenticated requests.

> Retransmitting the 401 (Unauthorized) or 407 (Proxy Authentication Required) status response amplifies the problem of an attacker using a falsified header field value (such as Via) to direct traffic to a third party.

In summary, the mutual authentication of proxy servers through mechanisms such as TLS significantly reduces the potential for rogue intermediaries to introduce falsified requests or responses that can deny service. This commensurately makes it harder for attackers to make innocent SIP nodes into agents of amplification.

26.4 Limitations

Although these security mechanisms, when applied in a judicious manner, can thwart many threats, there are limitations in the scope of the mechanisms that must be understood by implementers and network operators.

26.4.1 HTTP Digest

One of the primary limitations of using HTTP Digest in SIP is that the integrity mechanisms in Digest do not work very well for SIP. Specifically, they offer protection of the Request-URI and the method of a message, but not for any of the header fields that UAs would most likely wish to secure.

The existing replay protection mechanisms described in RFC 2617 also have some limitations for SIP. The next-nonce mechanism, for example, does not support pipelined requests. The nonce-count mechanism should be used for replay protection.

Another limitation of HTTP Digest is the scope of realms. Digest is valuable when a user wants to authenticate themselves to a resource with which they have a pre-existing association, like a service

provider of which the user is a customer (which is quite a common scenario and thus Digest provides an extremely useful function). By way of contrast, the scope of TLS is interdomain or multirealm, since certificates are often globally verifiable, so that the UA can authenticate the server with no pre-existing association.

26.4.2 S/MIME

The largest outstanding defect with the S/MIME mechanism is the lack of a prevalent public key infrastructure for end users. If self-signed certificates (or certificates that cannot be verified by one of the participants in a dialog) are used, the SIP-based key exchange mechanism described in Section 23.2 is susceptible to a man-in-the-middle attack with which an attacker can potentially inspect and modify S/MIME bodies. The attacker needs to intercept the first exchange of keys between the two parties in a dialog, remove the existing CMS-detached signatures from the request and response, and insert a different CMS-detached signature containing a certificate supplied by the attacker (but which seems to be a certificate for the proper address-of-record). Each party will think they have exchanged keys with the other, when in fact each has the public key of the attacker.

It is important to note that the attacker can only leverage this vulnerability on the first exchange of keys between two parties - on subsequent occasions, the alteration of the key would be noticeable to the UAs. It would also be difficult for the attacker to remain in the path of all future dialogs between the two parties over time (as potentially days, weeks, or years pass).

SSH is susceptible to the same man-in-the-middle attack on the first exchange of keys; however, it is widely acknowledged that while SSH is not perfect, it does improve the security of connections. The use of key fingerprints could provide some assistance to SIP, just as it does for SSH. For example, if two parties use SIP to establish a voice communications session, each could read off the fingerprint of the key they received from the other, which could be compared against the original. It would certainly be more difficult for the man-in-the-middle to emulate the voices of the participants than their signaling (a practice that was used with the Clipper chip-based secure telephone).

The S/MIME mechanism allows UAs to send encrypted requests without preamble if they possess a certificate for the destination address-of-record on their keyring. However, it is possible that any particular device registered for an address-of-record will not hold the certificate that has been previously employed by the device's current user, and that it will therefore be unable to process an

encrypted request properly, which could lead to some avoidable error signaling. This is especially likely when an encrypted request is forked.

The keys associated with S/MIME are most useful when associated with a particular user (an address-of-record) rather than a device (a UA). When users move between devices, it may be difficult to transport private keys securely between UAs; how such keys might be acquired by a device is outside the scope of this document.

Another, more prosaic difficulty with the S/MIME mechanism is that it can result in very large messages, especially when the SIP tunneling mechanism described in Section 23.4 is used. For that reason, it is RECOMMENDED that TCP should be used as a transport protocol when S/MIME tunneling is employed.

26.4.3 TLS

The most commonly voiced concern about TLS is that it cannot run over UDP; TLS requires a connection-oriented underlying transport protocol, which for the purposes of this document means TCP.

It may also be arduous for a local outbound proxy server and/or registrar to maintain many simultaneous long-lived TLS connections with numerous UAs. This introduces some valid scalability concerns, especially for intensive ciphersuites. Maintaining redundancy of long-lived TLS connections, especially when a UA is solely responsible for their establishment, could also be cumbersome.

TLS only allows SIP entities to authenticate servers to which they are adjacent; TLS offers strictly hop-by-hop security. Neither TLS, nor any other mechanism specified in this document, allows clients to authenticate proxy servers to whom they cannot form a direct TCP connection.

26.4.4 SIPS URIs

Actually using TLS on every segment of a request path entails that the terminating UAS must be reachable over TLS (perhaps registering with a SIPS URI as a contact address). This is the preferred use of SIPS. Many valid architectures, however, use TLS to secure part of the request path, but rely on some other mechanism for the final hop to a UAS, for example. Thus SIPS cannot guarantee that TLS usage will be truly end-to-end. Note that since many UAs will not accept incoming TLS connections, even those UAs that do support TLS may be required to maintain persistent TLS connections as described in the TLS limitations section above in order to receive requests over TLS as a UAS.

Location services are not required to provide a SIPS binding for a SIPS Request-URI. Although location services are commonly populated by user registrations (as described in Section 10.2.1), various other protocols and interfaces could conceivably supply contact addresses for an AOR, and these tools are free to map SIPS URIs to SIP URIs as appropriate. When queried for bindings, a location service returns its contact addresses without regard for whether it received a request with a SIPS Request-URI. If a redirect server is accessing the location service, it is up to the entity that processes the Contact header field of a redirection to determine the propriety of the contact addresses.

Ensuring that TLS will be used for all of the request segments up to the target domain is somewhat complex. It is possible that cryptographically authenticated proxy servers along the way that are non-compliant or compromised may choose to disregard the forwarding rules associated with SIPS (and the general forwarding rules in Section 16.6). Such malicious intermediaries could, for example, retarget a request from a SIPS URI to a SIP URI in an attempt to downgrade security.

Alternatively, an intermediary might legitimately retarget a request from a SIP to a SIPS URI. Recipients of a request whose Request-URI uses the SIPS URI scheme thus cannot assume on the basis of the Request-URI alone that SIPS was used for the entire request path (from the client onwards).

To address these concerns, it is RECOMMENDED that recipients of a request whose Request-URI contains a SIP or SIPS URI inspect the To header field value to see if it contains a SIPS URI (though note that it does not constitute a breach of security if this URI has the same scheme but is not equivalent to the URI in the To header field). Although clients may choose to populate the Request-URI and To header field of a request differently, when SIPS is used this disparity could be interpreted as a possible security violation, and the request could consequently be rejected by its recipient. Recipients MAY also inspect the Via header chain in order to double-check whether or not TLS was used for the entire request path until the local administrative domain was reached. S/MIME may also be used by the originating UAC to help ensure that the original form of the To header field is carried end-to-end.

If the UAS has reason to believe that the scheme of the Request-URI has been improperly modified in transit, the UA SHOULD notify its user of a potential security breach.

As a further measure to prevent downgrade attacks, entities that accept only SIPS requests MAY also refuse connections on insecure ports.

End users will undoubtedly discern the difference between SIPS and SIP URIs, and they may manually edit them in response to stimuli. This can either benefit or degrade security. For example, if an attacker corrupts a DNS cache, inserting a fake record set that effectively removes all SIPS records for a proxy server, then any SIPS requests that traverse this proxy server may fail. When a user, however, sees that repeated calls to a SIPS AOR are failing, they could on some devices manually convert the scheme from SIPS to SIP and retry. Of course, there are some safeguards against this (if the destination UA is truly paranoid it could refuse all non-SIPS requests), but it is a limitation worth noting. On the bright side, users might also divine that 'SIPS' would be valid even when they are presented only with a SIP URI.

26.5 Privacy

SIP messages frequently contain sensitive information about their senders - not just what they have to say, but with whom they communicate, when they communicate and for how long, and from where they participate in sessions. Many applications and their users require that this sort of private information be hidden from any parties that do not need to know it.

Note that there are also less direct ways in which private information can be divulged. If a user or service chooses to be reachable at an address that is guessable from the person's name and organizational affiliation (which describes most addresses-of-record), the traditional method of ensuring privacy by having an unlisted "phone number" is compromised. A user location service can infringe on the privacy of the recipient of a session invitation by divulging their specific whereabouts to the caller; an implementation consequently SHOULD be able to restrict, on a per-user basis, what kind of location and availability information is given out to certain classes of callers. This is a whole class of problem that is expected to be studied further in ongoing SIP work.

In some cases, users may want to conceal personal information in header fields that convey identity. This can apply not only to the From and related headers representing the originator of the request, but also the To - it may not be appropriate to convey to the final destination a speed-dialing nickname, or an unexpanded identifier for a group of targets, either of which would be removed from the Request-URI as the request is routed, but not changed in the To

header field if the two were initially identical. Thus it MAY be desirable for privacy reasons to create a To header field that differs from the Request-URI.

27 IANA Considerations

All method names, header field names, status codes, and option tags used in SIP applications are registered with IANA through instructions in an IANA Considerations section in an RFC.

The specification instructs the IANA to create four new sub-registries under http://www.iana.org/assignments/sip-parameters: Option Tags, Warning Codes (warn-codes), Methods and Response Codes, added to the sub-registry of Header Fields that is already present there.

27.1 Option Tags

This specification establishes the Option Tags sub-registry under http://www.iana.org/assignments/sip-parameters.

Option tags are used in header fields such as Require, Supported, Proxy-Require, and Unsupported in support of SIP compatibility mechanisms for extensions (Section 19.2). The option tag itself is a string that is associated with a particular SIP option (that is, an extension). It identifies the option to SIP endpoints.

Option tags are registered by the IANA when they are published in standards track RFCs. The IANA Considerations section of the RFC must include the following information, which appears in the IANA registry along with the RFC number of the publication.

- Name of the option tag. The name MAY be of any length, but SHOULD be no more than twenty characters long. The name MUST consist of alphanum (Section 25) characters only.

- Descriptive text that describes the extension.

27.2 Warn-Codes

This specification establishes the Warn-codes sub-registry under http://www.iana.org/assignments/sip-parameters and initiates its population with the warn-codes listed in Section 20.43. Additional warn-codes are registered by RFC publication.

The descriptive text for the table of warn-codes is:

Warning codes provide information supplemental to the status code in SIP response messages when the failure of the transaction results from a Session Description Protocol (SDP) (RFC 2327 [1]) problem.

The "warn-code" consists of three digits. A first digit of "3" indicates warnings specific to SIP. Until a future specification describes uses of warn-codes other than 3xx, only 3xx warn-codes may be registered.

Warnings 300 through 329 are reserved for indicating problems with keywords in the session description, 330 through 339 are warnings related to basic network services requested in the session description, 370 through 379 are warnings related to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

27.3 Header Field Names

This obsoletes the IANA instructions about the header sub-registry under http://www.iana.org/assignments/sip-parameters.

The following information needs to be provided in an RFC publication in order to register a new header field name:

- o The RFC number in which the header is registered;
- o the name of the header field being registered;
- o a compact form version for that header field, if one is defined;

Some common and widely used header fields MAY be assigned one-letter compact forms (Section 7.3.3). Compact forms can only be assigned after SIP working group review, followed by RFC publication.

27.4 Method and Response Codes

This specification establishes the Method and Response-Code sub-registries under http://www.iana.org/assignments/sip-parameters and initiates their population as follows. The initial Methods table is:

```
INVITE                 [RFC3261]
ACK                    [RFC3261]
BYE                    [RFC3261]
CANCEL                 [RFC3261]
REGISTER               [RFC3261]
OPTIONS                [RFC3261]
INFO                   [RFC2976]
```

The response code table is initially populated from Section 21, the portions labeled Informational, Success, Redirection, Client-Error, Server-Error, and Global-Failure. The table has the following format:

```
Type (e.g., Informational)
      Number    Default Reason Phrase         [RFC3261]
```

The following information needs to be provided in an RFC publication in order to register a new response code or method:

- o The RFC number in which the method or response code is registered;
- o the number of the response code or name of the method being registered;
- o the default reason phrase for that response code, if applicable;

27.5 The "message/sip" MIME type.

This document registers the "message/sip" MIME media type in order to allow SIP messages to be tunneled as bodies within SIP, primarily for end-to-end security purposes. This media type is defined by the following information:

Media type name: message
Media subtype name: sip
Required parameters: none

Optional parameters: version
version: The SIP-Version number of the enclosed message (e.g., "2.0"). If not present, the version defaults to "2.0".
Encoding scheme: SIP messages consist of an 8-bit header optionally followed by a binary MIME data object. As such, SIP messages must be treated as binary. Under normal circumstances SIP messages are transported over binary-capable transports, no special encodings are needed.

Security considerations: see below
Motivation and examples of this usage as a security mechanism in concert with S/MIME are given in 23.4.

27.6 New Content-Disposition Parameter Registrations

This document also registers four new Content-Disposition header "disposition-types": alert, icon, session and render. The authors request that these values be recorded in the IANA registry for Content-Dispositions.

Descriptions of these "disposition-types", including motivation and examples, are given in Section 20.11.

Short descriptions suitable for the IANA registry are:

```
alert     the body is a custom ring tone to alert the user
icon      the body is displayed as an icon to the user
render    the body should be displayed to the user
session   the body describes a communications session, for
          example, as RFC 2327 SDP body
```

28 Changes From RFC 2543

This RFC revises RFC 2543. It is mostly backwards compatible with RFC 2543. The changes described here fix many errors discovered in RFC 2543 and provide information on scenarios not detailed in RFC 2543. The protocol has been presented in a more cleanly layered model here.

We break the differences into functional behavior that is a substantial change from RFC 2543, which has impact on interoperability or correct operation in some cases, and functional behavior that is different from RFC 2543 but not a potential source of interoperability problems. There have been countless clarifications as well, which are not documented here.

28.1 Major Functional Changes

- o When a UAC wishes to terminate a call before it has been answered, it sends CANCEL. If the original INVITE still returns a 2xx, the UAC then sends BYE. BYE can only be sent on an existing call leg (now called a dialog in this RFC), whereas it could be sent at any time in RFC 2543.

- o The SIP BNF was converted to be RFC 2234 compliant.

- o SIP URL BNF was made more general, allowing a greater set of characters in the user part. Furthermore, comparison rules were simplified to be primarily case-insensitive, and detailed handling of comparison in the presence of parameters was described. The most substantial change is that a URI with a parameter with the default value does not match a URI without that parameter.

- o Removed Via hiding. It had serious trust issues, since it relied on the next hop to perform the obfuscation process. Instead, Via hiding can be done as a local implementation choice in stateful proxies, and thus is no longer documented.

- o In RFC 2543, CANCEL and INVITE transactions were intermingled. They are separated now. When a user sends an INVITE and then a CANCEL, the INVITE transaction still terminates normally. A UAS needs to respond to the original INVITE request with a 487 response.

- o Similarly, CANCEL and BYE transactions were intermingled; RFC 2543 allowed the UAS not to send a response to INVITE when a BYE was received. That is disallowed here. The original INVITE needs a response.

- o In RFC 2543, UAs needed to support only UDP. In this RFC, UAs need to support both UDP and TCP.

- o In RFC 2543, a forking proxy only passed up one challenge from downstream elements in the event of multiple challenges. In this RFC, proxies are supposed to collect all challenges and place them into the forwarded response.

- o In Digest credentials, the URI needs to be quoted; this is unclear from RFC 2617 and RFC 2069 which are both inconsistent on it.

- o SDP processing has been split off into a separate specification [13], and more fully specified as a formal offer/answer exchange process that is effectively tunneled through SIP. SDP is allowed in INVITE/200 or 200/ACK for baseline SIP implementations; RFC 2543 alluded to the ability to use it in INVITE, 200, and ACK in a single transaction, but this was not well specified. More complex SDP usages are allowed in extensions.

- Added full support for IPv6 in URIs and in the Via header field. Support for IPv6 in Via has required that its header field parameters allow the square bracket and colon characters. These characters were previously not permitted. In theory, this could cause interop problems with older implementations. However, we have observed that most implementations accept any non-control ASCII character in these parameters.

- DNS SRV procedure is now documented in a separate specification [4]. This procedure uses both SRV and NAPTR resource records and no longer combines data from across SRV records as described in RFC 2543.

- Loop detection has been made optional, supplanted by a mandatory usage of Max-Forwards. The loop detection procedure in RFC 2543 had a serious bug which would report "spirals" as an error condition when it was not. The optional loop detection procedure is more fully and correctly specified here.

- Usage of tags is now mandatory (they were optional in RFC 2543), as they are now the fundamental building blocks of dialog identification.

- Added the Supported header field, allowing for clients to indicate what extensions are supported to a server, which can apply those extensions to the response, and indicate their usage with a Require in the response.

- Extension parameters were missing from the BNF for several header fields, and they have been added.

- Handling of Route and Record-Route construction was very underspecified in RFC 2543, and also not the right approach. It has been substantially reworked in this specification (and made vastly simpler), and this is arguably the largest change. Backwards compatibility is still provided for deployments that do not use "pre-loaded routes", where the initial request has a set of Route header field values obtained in some way outside of Record-Route. In those situations, the new mechanism is not interoperable.

- In RFC 2543, lines in a message could be terminated with CR, LF, or CRLF. This specification only allows CRLF.

- o Usage of Route in CANCEL and ACK was not well defined in RFC 2543. It is now well specified; if a request had a Route header field, its CANCEL or ACK for a non-2xx response to the request need to carry the same Route header field values. ACKs for 2xx responses use the Route values learned from the Record-Route of the 2xx responses.

- o RFC 2543 allowed multiple requests in a single UDP packet. This usage has been removed.

- o Usage of absolute time in the Expires header field and parameter has been removed. It caused interoperability problems in elements that were not time synchronized, a common occurrence. Relative times are used instead.

- o The branch parameter of the Via header field value is now mandatory for all elements to use. It now plays the role of a unique transaction identifier. This avoids the complex and bug-laden transaction identification rules from RFC 2543. A magic cookie is used in the parameter value to determine if the previous hop has made the parameter globally unique, and comparison falls back to the old rules when it is not present. Thus, interoperability is assured.

- o In RFC 2543, closure of a TCP connection was made equivalent to a CANCEL. This was nearly impossible to implement (and wrong) for TCP connections between proxies. This has been eliminated, so that there is no coupling between TCP connection state and SIP processing.

- o RFC 2543 was silent on whether a UA could initiate a new transaction to a peer while another was in progress. That is now specified here. It is allowed for non-INVITE requests, disallowed for INVITE.

- o PGP was removed. It was not sufficiently specified, and not compatible with the more complete PGP MIME. It was replaced with S/MIME.

- o Added the "sips" URI scheme for end-to-end TLS. This scheme is not backwards compatible with RFC 2543. Existing elements that receive a request with a SIPS URI scheme in the Request-URI will likely reject the request. This is actually a feature; it ensures that a call to a SIPS URI is only delivered if all path hops can be secured.

- o Additional security features were added with TLS, and these are described in a much larger and complete security considerations section.

- o In RFC 2543, a proxy was not required to forward provisional responses from 101 to 199 upstream. This was changed to MUST. This is important, since many subsequent features depend on delivery of all provisional responses from 101 to 199.

- o Little was said about the 503 response code in RFC 2543. It has since found substantial use in indicating failure or overload conditions in proxies. This requires somewhat special treatment. Specifically, receipt of a 503 should trigger an attempt to contact the next element in the result of a DNS SRV lookup. Also, 503 response is only forwarded upstream by a proxy under certain conditions.

- o RFC 2543 defined, but did no sufficiently specify, a mechanism for UA authentication of a server. That has been removed. Instead, the mutual authentication procedures of RFC 2617 are allowed.

- o A UA cannot send a BYE for a call until it has received an ACK for the initial INVITE. This was allowed in RFC 2543 but leads to a potential race condition.

- o A UA or proxy cannot send CANCEL for a transaction until it gets a provisional response for the request. This was allowed in RFC 2543 but leads to potential race conditions.

- o The action parameter in registrations has been deprecated. It was insufficient for any useful services, and caused conflicts when application processing was applied in proxies.

- o RFC 2543 had a number of special cases for multicast. For example, certain responses were suppressed, timers were adjusted, and so on. Multicast now plays a more limited role, and the protocol operation is unaffected by usage of multicast as opposed to unicast. The limitations as a result of that are documented.

- o Basic authentication has been removed entirely and its usage forbidden.

- o Proxies no longer forward a 6xx immediately on receiving it. Instead, they CANCEL pending branches immediately. This avoids a potential race condition that would result in a UAC getting a 6xx followed by a 2xx. In all cases except this race condition, the result will be the same - the 6xx is forwarded upstream.

- o RFC 2543 did not address the problem of request merging. This occurs when a request forks at a proxy and later rejoins at an element. Handling of merging is done only at a UA, and procedures are defined for rejecting all but the first request.

28.2 Minor Functional Changes

- o Added the Alert-Info, Error-Info, and Call-Info header fields for optional content presentation to users.

- o Added the Content-Language, Content-Disposition and MIME-Version header fields.

- o Added a "glare handling" mechanism to deal with the case where both parties send each other a re-INVITE simultaneously. It uses the new 491 (Request Pending) error code.

- o Added the In-Reply-To and Reply-To header fields for supporting the return of missed calls or messages at a later time.

- o Added TLS and SCTP as valid SIP transports.

- o There were a variety of mechanisms described for handling failures at any time during a call; those are now generally unified. BYE is sent to terminate.

- o RFC 2543 mandated retransmission of INVITE responses over TCP, but noted it was really only needed for 2xx. That was an artifact of insufficient protocol layering. With a more coherent transaction layer defined here, that is no longer needed. Only 2xx responses to INVITEs are retransmitted over TCP.

- o Client and server transaction machines are now driven based on timeouts rather than retransmit counts. This allows the state machines to be properly specified for TCP and UDP.

- o The Date header field is used in REGISTER responses to provide a simple means for auto-configuration of dates in user agents.

- o Allowed a registrar to reject registrations with expirations that are too short in duration. Defined the 423 response code and the Min-Expires for this purpose.

29 Normative References

[1] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers", RFC 3263, June 2002.

[5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[9] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April 2000.

[10] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", RFC 3264, June 2002.

[14] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[15] Postel, J., "DoD Standard Transmission Control Protocol", RFC 761, January 1980.

[16] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[17] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[19] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F., Watson, M. and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC 3204, December 2001.

[20] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[21] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[23] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[24] Ramsdell B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[26] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

30 Informative References

[27] R. Pandya, "Emerging mobile and personal communication systems," IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.

[28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November 2000.

[31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

[34] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[37] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000.

[38] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.

[39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.

[40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis, D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call Flow Examples", Work in Progress.

[41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience, Vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.

[42] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar. 1996.

[43] Floyd, S., "Congestion Control Principles", RFC 2914, September 2000.

A Table of Timer Values

Table 4 summarizes the meaning and defaults of the various timers used by this specification.

Timer	Value	Section	Meaning
T1	500ms default	Section 17.1.1.1	RTT Estimate
T2	4s	Section 17.1.2.2	The maximum retransmit interval for non-INVITE requests and INVITE responses
T4	5s	Section 17.1.2.2	Maximum duration a message will remain in the network
Timer A	initially T1	Section 17.1.1.2	INVITE request retransmit interval, for UDP only
Timer B	64*T1	Section 17.1.1.2	INVITE transaction timeout timer
Timer C	> 3min	Section 16.6 bullet 11	proxy INVITE transaction timeout
Timer D	> 32s for UDP 0s for TCP/SCTP	Section 17.1.1.2	Wait time for response retransmits
Timer E	initially T1	Section 17.1.2.2	non-INVITE request retransmit interval, UDP only
Timer F	64*T1	Section 17.1.2.2	non-INVITE transaction timeout timer
Timer G	initially T1	Section 17.2.1	INVITE response retransmit interval
Timer H	64*T1	Section 17.2.1	Wait time for ACK receipt
Timer I	T4 for UDP 0s for TCP/SCTP	Section 17.2.1	Wait time for ACK retransmits
Timer J	64*T1 for UDP 0s for TCP/SCTP	Section 17.2.2	Wait time for non-INVITE request retransmits
Timer K	T4 for UDP 0s for TCP/SCTP	Section 17.1.2.2	Wait time for response retransmits

Table 4: Summary of timers

Acknowledgments

We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions. Detailed comments were provided by Ofir Arkin, Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan, Keith Drage, Bill Fenner, Cedric Fluckiger, Yaron Goland, John Hearty, Bernie Hoeneisen, Jo Hornsby, Phil Hoffer, Christian Huitema, Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kristensen, Jonathan Lennox, Gethin Liddell, Allison Mankin, William Marshall, Rohan Mahy, Keith Moore, Vern Paxson, Bob Penfield, Moshe J. Sambol, Chip Sharp, Igor Slepchin, Eric Tremblay, and Rick Workman.

Brian Rosen provided the compiled BNF.

Jean Mahoney provided technical writing assistance.

This work is based, inter alia, on [41,42].

Authors' Addresses

Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of RFC 2543. All listed authors actively contributed large amounts of text to this document.

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Ave
East Hanover, NJ 07936
USA

EMail: jdrosen@dynamicsoft.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA

EMail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland

EMail: Gonzalo.Camarillo@ericsson.com

Alan Johnston
WorldCom
100 South 4th Street
St. Louis, MO 63102
USA

EMail: alan.johnston@wcom.com

Jon Peterson
NeuStar, Inc
1800 Sutter Street, Suite 570
Concord, CA 94520
USA

EMail: jon.peterson@neustar.com

Robert Sparks
dynamicsoft, Inc.
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
USA

EMail: rsparks@dynamicsoft.com

Mark Handley
International Computer Science Institute
1947 Center St, Suite 600
Berkeley, CA 94704
USA

EMail: mjh@icir.org

Eve Schooler
AT&T Labs-Research
75 Willow Road
Menlo Park, CA 94025
USA

EMail: schooler@research.att.com

Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

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

F. Andreasen
B. Foster
Cisco Systems

January 2003

Media Gateway Control Protocol (MGCP) Version 1.0

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

IESG Note

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

Abstract

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

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

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

Table of Contents

1. Introduction...5
1.1 Relation with the H.323 Standards..........................7
1.2 Relation with the IETF Standards...........................8
1.3 Definitions..9
1.4 Conventions used in this Document..........................9
2. Media Gateway Control Interface............................10
2.1 Model and Naming Conventions..............................10
2.1.1 Types of Endpoints......................................10
2.1.2 Endpoint Identifiers....................................14
2.1.3 Calls and Connections...................................16
2.1.4 Names of Call Agents and Other Entities.................22
2.1.5 Digit Maps..23
2.1.6 Packages..26
2.1.7 Events and Signals......................................28
2.2 Usage of SDP..33
2.3 Gateway Control Commands..................................33
2.3.1 Overview of Commands....................................33
2.3.2 EndpointConfiguration...................................36
2.3.3 NotificationRequest.....................................37
2.3.4 Notify..44
2.3.5 CreateConnection..46
2.3.6 ModifyConnection..52
2.3.7 DeleteConnection (from the Call Agent)..................54
2.3.8 DeleteConnection (from the gateway).....................58
2.3.9 DeleteConnection (multiple connections from the Call Agent) 59
2.3.10 AuditEndpoint..60
2.3.11 AuditConnection..65
2.3.12 RestartInProgress......................................66
2.4 Return Codes and Error Codes..............................69
2.5 Reason Codes..74
2.6 Use of Local Connection Options and Connection Descriptors..75
2.7 Resource Reservations.....................................77
3. Media Gateway Control Protocol.............................77
3.1 General Description.......................................78
3.2 Command Header..79
3.2.1 Command Line..79
3.2.2 Parameter Lines...82
3.3 Format of response headers...............................101
3.3.1 CreateConnection Response..............................104
3.3.2 ModifyConnection Response..............................105

3.3.3 DeleteConnection Response....................................106
3.3.4 NotificationRequest Response..................................106
3.3.5 Notify Response...106
3.3.6 AuditEndpoint Response..106
3.3.7 AuditConnection Response......................................107
3.3.8 RestartInProgress Response....................................108
3.4 Encoding of the Session Description (SDP)..................108
3.4.1 Usage of SDP for an Audio Service.............................110
3.4.2 Usage of SDP for LOCAL Connections............................110
3.5 Transmission over UDP...111
3.5.1 Providing the At-Most-Once Functionality......................112
3.5.2 Transaction Identifiers and Three Ways Handshake...........113
3.5.3 Computing Retransmission Timers...............................114
3.5.4 Maximum Datagram Size, Fragmentation and Reassembly........115
3.5.5 Piggybacking...116
3.5.6 Provisional Responses...117
4. States, Failover and Race Conditions.........................119
4.1 Failover Assumptions and Highlights..........................119
4.2 Communicating with Gateways..................................121
4.3 Retransmission, and Detection of Lost Associations:........122
4.4 Race Conditions...126
4.4.1 Quarantine List...127
4.4.2 Explicit Detection..133
4.4.3 Transactional Semantics.......................................134
4.4.4 Ordering of Commands, and Treatment of Misorder............135
4.4.5 Endpoint Service States.......................................137
4.4.6 Fighting the Restart Avalanche...............................140
4.4.7 Disconnected Endpoints..143
4.4.8 Load Control in General.......................................146
5. Security Requirements...147
5.1 Protection of Media Connections..............................148
6. Packages...148
6.1 Actions..150
6.2 BearerInformation..150
6.3 ConnectionModes..151
6.4 ConnectionParameters...151
6.5 DigitMapLetters..151
6.6 Events and Signals...152
6.6.1 Default and Reserved Events..................................155
6.7 ExtensionParameters..156
6.8 LocalConnectionOptions.......................................157
6.9 Reason Codes...157
6.10 RestartMethods...158
6.11 Return Codes...158
7. Versions and Compatibility...................................158
7.1 Changes from RFC 2705..158
8. Security Considerations.......................................164
9. Acknowledgments..164

10. References..164
Appendix A: Formal Syntax Description of the Protocol.............167
Appendix B: Base Package..175
B.1 Events..175
B.2 Extension Parameters..176
B.2.1 PersistentEvents..176
B.2.2 NotificationState...177
B.3 Verbs...177
Appendix C: IANA Considerations...................................179
C.1 New MGCP Package Sub-Registry...............................179
C.2 New MGCP Package..179
C.3 New MGCP LocalConnectionOptions Sub-Registry................179
Appendix D: Mode Interactions.....................................180
Appendix E: Endpoint Naming Conventions...........................182
E.1 Analog Access Line Endpoints................................182
E.2 Digital Trunks..182
E.3 Virtual Endpoints...183
E.4 Media Gateway...184
E.5 Range Wildcards...184
Appendix F: Example Command Encodings.............................185
F.1 NotificationRequest...185
F.2 Notify..186
F.3 CreateConnection..186
F.4 ModifyConnection..189
F.5 DeleteConnection (from the Call Agent)......................189
F.6 DeleteConnection (from the gateway).........................190
F.7 DeleteConnection (multiple connections
from the Call Agent)..190
F.8 AuditEndpoint...191
F.9 AuditConnection...192
F.10 RestartInProgress..193
Appendix G: Example Call Flows....................................194
G.1 Restart...195
G.1.1 Residential Gateway Restart...............................195
G.1.2 Call Agent Restart..198
G.2 Connection Creation...200
G.2.1 Residential Gateway to Residential Gateway................200
G.3 Connection Deletion...206
G.3.1 Residential Gateway to Residential Gateway................206
Authors' Addresses..209
Full Copyright Statement..210

1. Introduction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.1 Relation with the H.323 Standards

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

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

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

1.2 Relation with the IETF Standards

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

* the Session Description Protocol (SDP), RFC 2327

* the Session Announcement Protocol (SAP), RFC 2974

* the Session Initiation Protocol (SIP), RFC 3261

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

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

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

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

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

1.3 Definitions

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

1.4 Conventions used in this Document

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

2. Media Gateway Control Interface

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

2.1 Model and Naming Conventions

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

2.1.1 Types of Endpoints

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

* Digital channel (DS0),

* Analog line,

* Announcement server access point,

* Interactive Voice Response access point,

* Conference bridge access point,

* Packet relay,

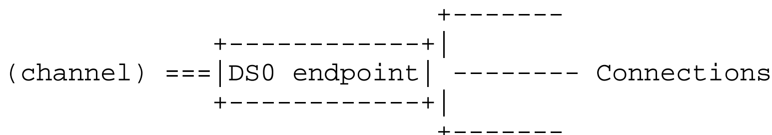
* ATM "trunk side" interface.

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

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

2.1.1.1 Digital Channel (DS0)

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

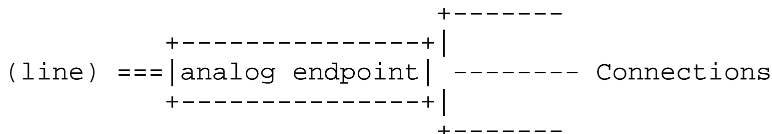


Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document.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

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

2.1.1.2 Analog Line

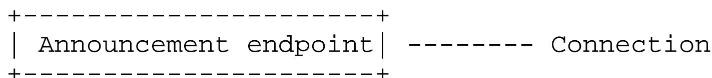
Analog lines can be used either as a "client" interface, providing service to a classic telephone unit, or as a "service" interface, allowing the gateway to send and receive analog calls. When the media gateway also supports a NAS service, the gateway shall be capable of receiving audio-encoded data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



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

2.1.1.3 Announcement Server Access Point

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

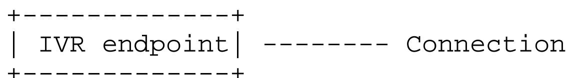


A given announcement endpoint is not expected to support more than one connection at a time. If several connections were established to the same endpoint, then the same announcements would be played simultaneously over all the connections.

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

2.1.1.4 Interactive Voice Response Access Point

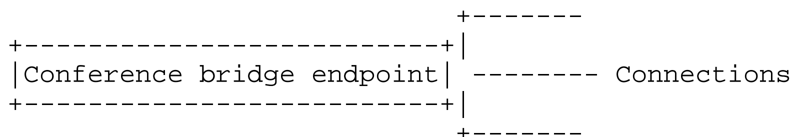
An Interactive Voice Response (IVR) endpoint provides access to an IVR service. Under requests from the Call Agent, the IVR server will "play" announcements and tones, and will "listen" to responses, such as DTMF input or voice messages, from the user. The requests from the Call Agent will follow the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



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

2.1.1.5 Conference Bridge Access Point

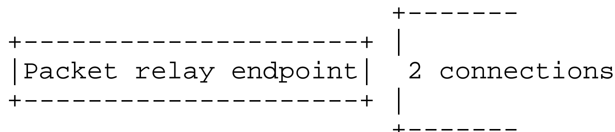
A conference bridge endpoint is used to provide access to a specific conference.



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

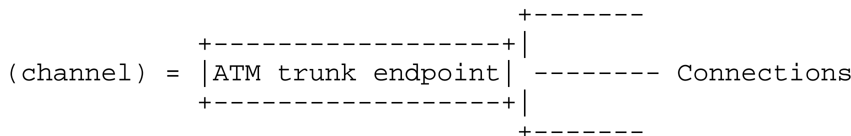
2.1.1.6 Packet Relay

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



2.1.1.7 ATM "trunk side" Interface

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



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

2.1.2 Endpoint Identifiers

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

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

* a local name within that gateway

Endpoint names are of the form:

    local-endpoint-name@domain-name

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

    mygateway.whatever.net

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

```
[192.168.1.2]
```

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

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

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

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

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

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

   ```
   term1/term2/term3
   ```

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

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

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

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

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

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

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

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

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

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

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

2.1.3 Calls and Connections

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

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

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

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

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

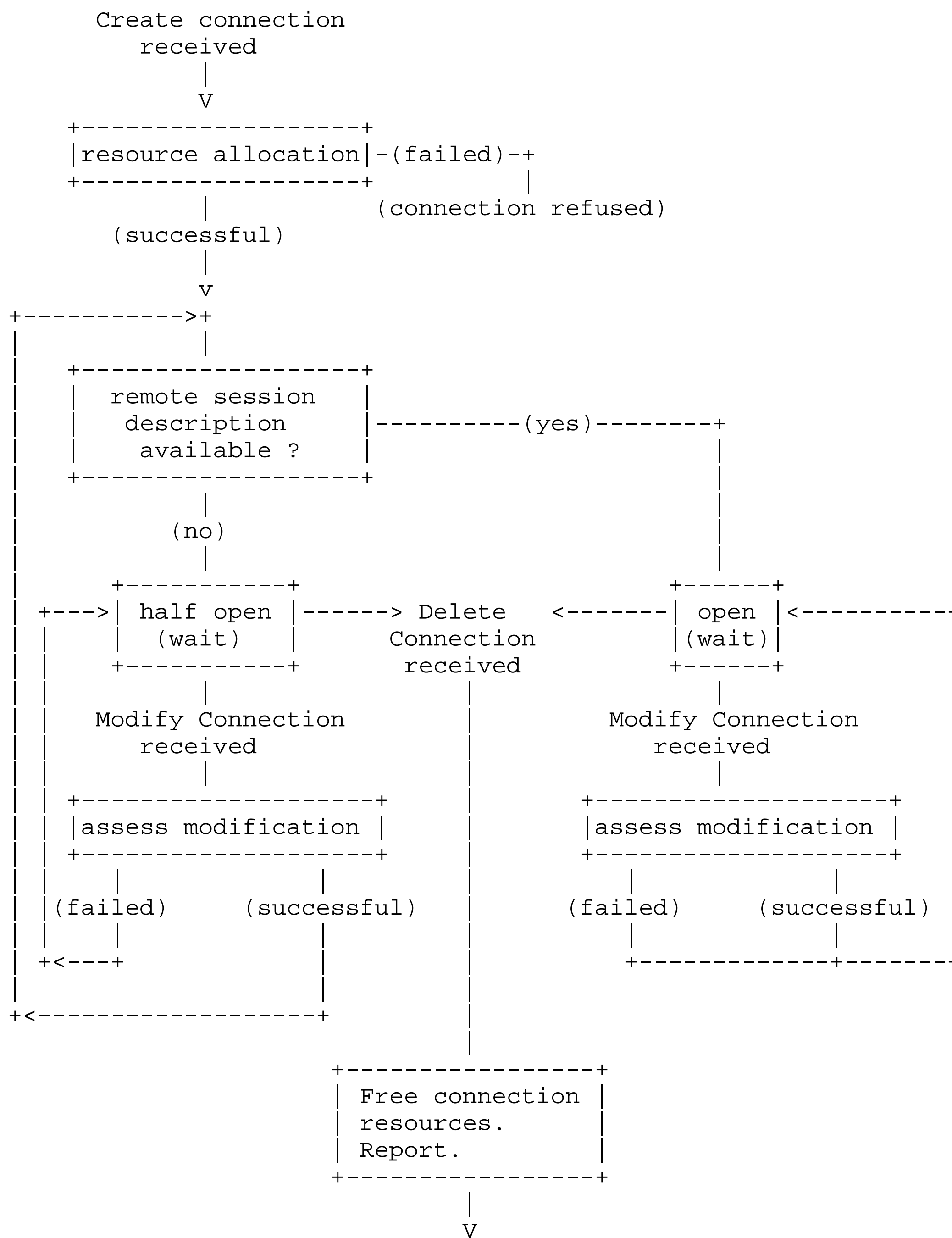
3) The Call Agent then uses a "modify connection" command to provide this second "session description" to the first endpoint. Once this is done, communication can proceed in both directions.

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

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

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

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


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

2.1.3.1 Names of Calls

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

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

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

2.1.3.2 Names of Connections

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

2.1.3.3 Management of Resources, Attributes of Connections

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

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

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

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

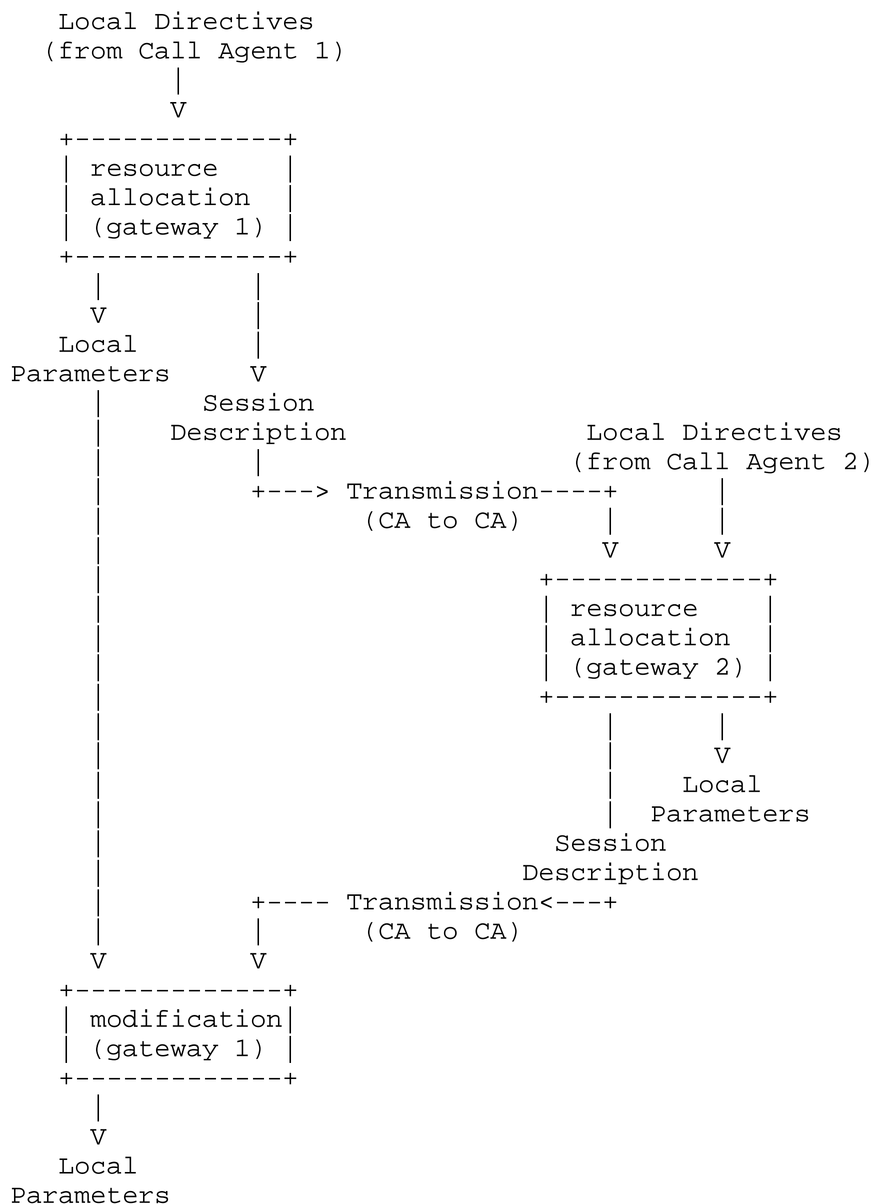
1) The local directives instruct the gateway on the choice of resources that should be used for a connection,

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

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

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

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



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

2.1.3.4 Special Case of Local Connections

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

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

* Connecting a call to a Conferencing unit,

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

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

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

2.1.4 Names of Call Agents and Other Entities

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

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

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

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

Reliability can be improved by using the following procedures:

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

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

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

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

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

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

2.1.5 Digit Maps

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
(xxxxxxx|x11)
```

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

The following digit map example is more subtle:

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

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

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

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

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

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

2.1.6 Packages

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

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

* BearerInformation

* LocalConnectionOptions

* ExtensionParameters

* ConnectionModes

* Events

* Signals

* Actions

* DigitMapLetters

* ConnectionParameters

* RestartMethods

* ReasonCodes

* Return codes

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

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

    package-name/extension

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

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

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

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

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

2.1.7 Events and Signals

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

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

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

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

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

L/dl

and

dl

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
R/qa@*
```

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

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

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

```
G/rt@$
```

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

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

```
*/all@*
```

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

Signals are divided into different types depending on their behavior:

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

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

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

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

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

2.2 Usage of SDP

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

2.3 Gateway Control Commands

2.3.1 Overview of Commands

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

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

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

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

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

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

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

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

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

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

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

* A specific announcement handled by an announcement server.

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.2 EndpointConfiguration

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

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

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

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

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

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

* otherwise, the BearerInformation parameter is OPTIONAL.

When omitted, BearerInformation MUST retain its current value.

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

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

2.3.3 NotificationRequest

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

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

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

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

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

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

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

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

* Swap audio,

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

* Accumulate according to Digit Map,

* Keep Signal(s) active,

* Process the Embedded Notification Request,

* Ignore the event.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

* Ringing,

* Busy tone,

* Call waiting tone,

* Off hook warning tone,

* Ringback tones on a connection.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.4 Notify

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

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

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

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

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

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

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

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

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

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

2.3.5 CreateConnection

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

* ask the residential gateway to start ringing,

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

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

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

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

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

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

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

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

2.3.6 ModifyConnection

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.7 DeleteConnection (from the Call Agent)

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

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

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

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

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

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

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

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

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

Number of packets sent:

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

Number of octets sent:

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

Number of packets received:

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

Number of octets received:

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

Number of packets lost:

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

Interarrival jitter:

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

Average transmission delay:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3.8 DeleteConnection (from the gateway)

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

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

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

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

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

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

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

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

2.3.9 DeleteConnection (multiple connections from the Call Agent)

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

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

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

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

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

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

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

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

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

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

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

2.3.10 AuditEndpoint

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  - Security: Whether media encryption is supported or not.

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

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

  - Modes: A list of supported connection modes.

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

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

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

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

2.3.11 AuditConnection

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

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

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

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

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

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

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

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

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

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

* Mode, the current mode of the connection.

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

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

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

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

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

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

2.3.12 RestartInProgress

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

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

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

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

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

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

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

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

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

The list of restart methods may be extended.

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

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

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

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

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

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

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

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

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

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

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

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

2.4 Return Codes and Error Codes

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

* values between 000 and 099 indicate a response acknowledgement

* values between 100 and 199 indicate a provisional response

* values between 200 and 299 indicate a successful completion

* values between 400 and 499 indicate a transient error

* values between 500 and 599 indicate a permanent error

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

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

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

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

The values that have been already defined are the following:

000 Response Acknowledgement.

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

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

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

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

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

401 The phone is already off hook.

402 The phone is already on hook.

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

404 Insufficient bandwidth at this time.

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

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

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

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

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

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

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

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

503 "All of" wildcard too complicated.

504 Unknown or unsupported command.

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

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

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

508 Unknown or unsupported quarantine handling.

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

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

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

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

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

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

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

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

517 Unsupported or invalid mode.

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

519 Endpoint does not have a digit map.

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

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

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

523 Unknown action or illegal combination of actions.

524 Internal inconsistency in LocalConnectionOptions.

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

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

527 Missing RemoteConnectionDescriptor.

528 Incompatible protocol version.

529 Internal hardware failure.

530 CAS signaling protocol error.

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

532 Unsupported value(s) in LocalConnectionOptions.

533 Response too large.

534 Codec negotiation failure.

535 Packetization period not supported.

536 Unknown or unsupported RestartMethod.

537 Unknown or unsupported digit map extension.

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

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

540 Per endpoint connection limit exceeded.

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

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

* Unknown 0xx code treated as 000.

* Unknown 1xx code treated as 100.

* Unknown 2xx code treated as 200.

* Unknown 3xx code treated as 521.

* Unknown 4xx code treated as 400.

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

2.5 Reason Codes

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

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

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

900 Endpoint malfunctioning.

901 Endpoint taken out-of-service.

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

903 QoS resource reservation was lost.

904 Manual intervention.

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

The set of reason codes can be extended.

2.6 Use of Local Connection Options and Connection Descriptors

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.7 Resource Reservations

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

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

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

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

3. Media Gateway Control Protocol

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

* EndpointConfiguration

* CreateConnection

* ModifyConnection

* DeleteConnection

* NotificationRequest

* Notify

* AuditEndpoint

* AuditConnection

* RestartInProgress

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

3.1 General Description

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

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

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

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

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

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

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

3.2 Command Header

The command header is composed of:

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

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

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

3.2.1 Command Line

The command line is composed of:

* The name of the requested verb,

* The identification of the transaction,

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

* The protocol version.

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

3.2.1.1 Coding of the Requested Verb

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

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

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

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

3.2.1.2 Transaction Identifiers

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

* a transaction identifier name space for sending transactions, and

* a transaction identifier name space for receiving transactions.

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

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

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

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

3.2.1.3 Coding of the Endpoint Identifiers and Entity Names

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

Examples of such addresses are:

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

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

Call-agent@ca.example.net:5234

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

3.2.1.4 Coding of the Protocol Version

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

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

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

```
MGCP 1.0
```

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

3.2.2 Parameter Lines

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

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

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

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

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

Notes (*):

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

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

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

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

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

* Package Extension Parameters (preferred)

* Vendor Extension Parameters

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

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

* a separate name space for the parameters.

* a convenient grouping of the extensions.

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

The package extension mechanism is the preferred extension method.

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

```
X-Flower: Daisy
```

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

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

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

3.2.2.1 BearerInformation

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

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

An example of bearer information encoding is:

```
B: e:mu
```

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

3.2.2.2 CallId

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

3.2.2.3 Capabilities

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

The parameters used are:

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

Packetization Period:
  A range may be specified.

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

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

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

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

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

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

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

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

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

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

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

An example capability is:

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

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

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

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

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

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

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

3.2.2.4 Coding of Event Names

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

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

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

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

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

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

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

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

The following are examples of event names:

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

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

The following are examples of such notations:

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

3.2.2.5 ConnectionId

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

3.2.2.6 ConnectionMode

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

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

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

The set of connection modes can be extended through packages.

3.2.2.7 ConnectionParameters

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

The connection parameter types are specified in the following table:

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

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

* Package Extension Parameters (preferred)

* Vendor Extension Parameters

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

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

* A separate name space for the parameters.

* A convenient grouping of the extensions.

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

The package extension mechanism is the preferred extension method.

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

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

An example of connection parameter encoding is:

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

3.2.2.8 DetectEvents

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

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

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

3.2.2.9 EventStates

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

```
ES: L/hu
```

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

3.2.2.10 LocalConnectionOptions

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

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

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

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

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

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

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

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

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

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

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

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

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

Examples of local connection options are:

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

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

* Package Extension attributes (preferred)

* Vendor Extension attributes

* Other Extension attributes

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

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

* A separate name space for the attributes.

* A convenient grouping of the extensions.

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

The package extension mechanism is the preferred extension method.

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

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

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

3.2.2.11 MaxMGCPDatagram

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

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

```
MD: 8100
```

3.2.2.12 ObservedEvents

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

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

3.2.2.13 PackageList

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

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

An example of a package list is:

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

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

3.2.2.14 QuarantineHandling

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

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

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

The following values are valid examples:

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

3.2.2.15 ReasonCode

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

```
E: 900 Endpoint malfunctioning
```

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

The set of reason codes can be extended through packages.

3.2.2.16 RequestedEvents

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2.17 RequestedInfo

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

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

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

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

```
F: A
```

3.2.2.18 RequestIdentifier

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

3.2.2.19 ResponseAck

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

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

An example of a response acknowledgement is:

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

3.2.2.20 RestartMethod

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

```
RM: restart
```

The set of restart methods can be extended through packages.

3.2.2.21 SignalRequests

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

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

* the name and parameters of the announcement,

* the string that should be displayed.

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

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

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

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

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

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

3.3 Format of response headers

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

An example of a response header could be:

```
200 1203 OK
```

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

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

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

Notes (*):

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

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

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

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

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

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

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

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

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

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

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

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

3.3.1 CreateConnection Response

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

```
200 1204 OK
I: FDE234C8

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

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

```
200 1204 OK
K:
I: FDE234C8

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

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

```
000 1204
```

3.3.2 ModifyConnection Response

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

```
200 1207 OK

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

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

```
200 1207 OK
K:
```

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

```
000 1207 OK
```

3.3.3 DeleteConnection Response

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

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

3.3.4 NotificationRequest Response

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

3.3.5 Notify Response

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

3.3.6 AuditEndpoint Response

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

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

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

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

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

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

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

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

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

3.3.7 AuditConnection Response

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

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

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

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

```
200 1203 OK

v=0
```

3.3.8 RestartInProgress Response

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

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

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

3.4 Encoding of the Session Description (SDP)

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

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

* Origin ("o="),

* Session Name ("s="), and

* Time active ("t=")

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

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

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

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

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

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

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

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

Session Name
s = <session name>

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

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

* The start time may be set to zero.

* The stop time should be set to zero.

Each of the three fields can be ignored upon reception.

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

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

3.4.1 Usage of SDP for an Audio Service

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

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

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

3.4.2 Usage of SDP for LOCAL Connections

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

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

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

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

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

An example of a LOCAL session description could be:

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

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

3.5 Transmission over UDP

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

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

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

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

3.5.1 Providing the At-Most-Once Functionality

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

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

The procedure uses a long timer value, noted T-HIST in the following. The timer MUST be set larger than the maximum duration of a transaction, which MUST take into account the maximum number of repetitions, the maximum value of the repetition timer and the maximum propagation delay of a packet in the network. A suggested value is 30 seconds.

The copy of the responses MAY be destroyed either T-HIST seconds after the response is issued, or when the gateway (or the Call Agent) receives a confirmation that the response has been received, through the "Response Acknowledgement". For transactions that are acknowledged through this attribute, the gateway SHALL keep a copy of the transaction-id (as opposed to the entire transaction response) for T-HIST seconds after the response is issued, in order to detect and ignore duplicate copies of the transaction request that could be produced by the network.

3.5.2 Transaction Identifiers and Three Ways Handshake

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

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

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

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

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

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

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

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

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

3.5.3 Computing Retransmission Timers

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

The specification purposely avoids specifying any value for the retransmission timers. These values are typically network dependent. The retransmission timers SHOULD normally estimate the timer by measuring the time spent between the sending of a command and the return of the first response to the command. At a minimum, a retransmission strategy involving exponential backoff MUST be implemented. One possibility is to use the algorithm implemented in TCP/IP, which uses two variables:

* the average acknowledgement delay, AAD, estimated through an exponentially smoothed average of the observed delays,

* the average deviation, ADEV, estimated through an exponentially smoothed average of the absolute value of the difference between the observed delay and the current average.

The retransmission timer, RTO, in TCP, is set to the sum of the average delay plus N times the average deviation, where N is a constant. In MGCP, the maximum value of the timer SHOULD however be bounded, in order to guarantee that no repeated packet will be received by the gateways after T-HIST seconds. A suggested maximum value for RTO (RTO-MAX) is 4 seconds. Implementers SHOULD consider bounding the minimum value of this timer as well [19].

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

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

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

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

This procedure has two effects. Because it includes an exponentially increasing component, it will automatically slow down the stream of messages in case of congestion. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

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

3.5.4 Maximum Datagram Size, Fragmentation and Reassembly

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

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

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

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

3.5.5 Piggybacking

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

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

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

Piggybacking can be used to achieve two things:

* Guaranteed in-order delivery and processing of messages.

* Fate sharing of message delivery.

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

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

3.5.6 Provisional Responses

Executing some transactions may require a long time. Long execution times may interact with the timer based retransmission procedure.

This may result either in an inordinate number of retransmissions, or in timer values that become too long to be efficient.

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

Gateways (or Call Agents) that start building up queues of transactions to be executed may send a provisional response with response code 101 to indicate this (see Section 4.4.8 for further details).

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

In order to reduce the delay in the system, it is RECOMMENDED to include a connection identifier and session description in a 100 provisional response to the CreateConnection command. If a session description would be returned by the ModifyConnection command, the session description SHOULD be included in the provisional response here as well. If the transaction completes successfully, the information returned in the provisional response MUST be repeated in the final response. It is considered a protocol error not to repeat this information or to change any of the previously supplied information in a successful response. If the transaction fails, an error code is returned - the information returned previously is no longer valid.

A currently executing CreateConnection or ModifyConnection transaction MUST be cancelled if a DeleteConnection command for the endpoint is received. In that case, a final response for the cancelled transaction SHOULD still be returned automatically (error code 407 - transaction aborted, is RECOMMENDED), and a final response for the cancelled transaction MUST be returned if a retransmission of the cancelled transaction is detected (see also Section 4.4.4).

MGCP entities that receive a provisional response SHALL switch to a longer repetition timer (LONGTRAN-TIMER) for that transaction. The purpose of this timer is primarily to detect processing failures. The default value of LONGTRAN-TIMER is 5 seconds, however the provisioning process may alter this. Note, that retransmissions MUST still satisfy the timing requirements specified in Section 3.5.1 and 3.5.3. Consequently LONGTRAN-TIMER MUST be smaller than T-HIST (it should in fact be considerably smaller). Also, entities MUST NOT let a transaction run forever. A transaction that is timed out by the entity SHOULD return error code 406 (transaction time-out). Per the definition of T-HIST (Section 3.5.1), the maximum transaction execution time is smaller than T-HIST (in a network with low delay, it can reasonably safely be approximated as T-HIST minus T-MAX), and a final response should be received no more than T-HIST seconds after the command was sent initially. Nevertheless, entities SHOULD wait for 2*T-HIST seconds before giving up on receiving a final response. Retransmission of the command MUST still cease after T-MAX seconds though. If a response is not received, the outcome of the transaction is not known. If the entity sending the command was a gateway, it now becomes "disconnected" and SHALL initiate the "disconnected" procedure (see Section 4.4.7).

When the transaction finishes execution, the final response is sent and the by now obsolete provisional response is deleted. In order to ensure rapid detection of a lost final response, final responses issued after provisional responses for a transaction SHOULD be acknowledged (unfortunately older RFC 2705 implementations may not do this, which is the only reason it is not an absolute requirement).

The endpoint SHOULD therefore include an empty "ResponseAck" parameter in those, and only those, final responses. The presence of the "ResponseAck" parameter in the final response SHOULD trigger a "Response Acknowledgement" response to be sent back to the endpoint. The Response Acknowledgement" response will then include the transaction-id of the response it acknowledges in the response header. Note that, for backwards compatibility, entities cannot depend on receiving such a "response acknowledgement", however it is strongly RECOMMENDED to support this behavior, as excessive delays in case of packet loss as well as excessive retransmissions may occur otherwise.

Receipt of a "Response Acknowledgement" response is subject to the same time-out and retransmission strategies and procedures as responses to commands, i.e., the sender of the final response will retransmit it if a "Response Acknowledgement" is not received in time. For backwards compatibility, failure to receive a "response acknowledgement" SHOULD NOT affect the roundtrip time estimates for subsequent commands, and furthermore MUST NOT lead to the endpoint becoming "disconnected". The "Response Acknowledgment" response is never acknowledged.

4. States, Failover and Race Conditions

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

4.1 Failover Assumptions and Highlights

The following protocol highlights are important to understanding Call Agent fail-over mechanisms:

* Call Agents are identified by their domain name (and optional port), not their network addresses, and several addresses can be associated with a domain name.

* An endpoint has one and only one Call Agent associated with it at any given point in time. The Call Agent associated with an endpoint is the current value of the "notified entity". The "notified entity" determines where the gateway will send it's commands. If the "notified entity" does not include a port number, the default Call Agent port number (2727) is assumed.

* NotifiedEntity is a parameter sent by the Call Agent to the gateway to set the "notified entity" for the endpoint.

* The "notified entity" for an endpoint is the last value of the NotifiedEntity parameter received for this endpoint. If no explicit NotifiedEntity parameter has ever been received, the "notified entity" defaults to a provisioned value. If no value was provisioned or an empty NotifiedEntity parameter was provided (both strongly discouraged) thereby making the "notified entity" empty, the "notified entity" is set to the source address of the last non-audit command for the endpoint. Thus auditing will not change the "notified entity".

* Responses to commands are sent to the source address of the command, regardless of the current "notified entity". When a Notify message needs to be piggybacked with the response, the datagram is still sent to the source address of the new command received, regardless of the current "notified entity".

The ability for the "notified entity" to resolve to multiple network addresses, allows a "notified entity" to represent a Call Agent with multiple physical interfaces on it and/or a logical Call Agent made up of multiple physical systems. The order of network addresses when a DNS name resolves to multiple addresses is non-deterministic so Call Agent fail-over schemes MUST NOT depend on any order (e.g., a gateway MUST be able to send a "Notify" to any of the resolved network addresses). On the other hand, the system is likely to be most efficient if the gateway sends commands to the interface with which it already has a current association. It is RECOMMENDED that gateways use the following algorithm to achieve that goal:

* If the "notified entity" resolves to multiple network addresses, and the source address of the request is one of those addresses, that network address is the preferred destination address for commands.

* If on the other hand, the source address of the request is not one of the resolved addresses, the gateway must choose one of the resolved addresses for commands.

* If the gateway fails to contact the network address chosen, it MUST try the alternatives in the resolved list as described in Section 4.3.

If an entire Call Agent becomes unavailable, the endpoints managed by that Call Agent will eventually become "disconnected". The only way for these endpoints to become connected again is either for the failed Call Agent to become available, or for a backup call agent to contact the affected endpoints with a new "notified entity".

When a backup Call Agent has taken over control of a group of endpoints, it is assumed that the failed Call Agent will communicate and synchronize with the backup Call Agent in order to transfer control of the affected endpoints back to the original Call Agent. Alternatively, the failed Call Agent could simply become the backup Call Agent.

We should note that handover conflict resolution between separate CA's is not in place - we are relying strictly on the CA's knowing what they are doing and communicating with each other (although AuditEndpoint can be used to learn about the current "notified entity"). If this is not the case, unexpected behavior may occur.

Note that as mentioned earlier, the default "notified entity" is provisioned and may include both domain name and port. For small gateways, provisioning may be done on a per endpoint basis. For much larger gateways, a single provisioning element may be provided for multiple endpoints or even for the entire gateway itself. In either case, once the gateway powers up, each endpoint MUST have its own "notified entity", so provisioned values for an aggregation of endpoints MUST be copied to the "notified entity" for each endpoint in the aggregation before operation proceeds. Where possible, the RestartInProgress command on restart SHOULD be sent to the provisioned "notified entity" based on an aggregation that allows the "all of" wild-card to be used. This will reduce the number of RestartInProgress messages.

Another way of viewing the use of "notified entity" is in terms of associations between gateways and Call Agents. The "notified entity" is a means to set up that association, and governs where the gateway will send commands to. Commands received by the gateway however may come from any source. The association is initially provisioned with a provisioned "notified entity", so that on power up RestartInProgress and persistent events that occur prior to the first NotificationRequest from Call Agents will be sent to the provisioned Call Agent. Once a Call Agent makes a request, however it may include the NotifiedEntity parameter and set up a new association. Since the "notified entity" persists across calls, the association remains intact until a new "notified entity" is provided.

4.2 Communicating with Gateways

Endpoint names in gateways include a local name indicating the specific endpoint and a domain name indicating the host/gateway where the endpoint resides. Gateways may have several interfaces for redundancy.

In gateways that have routing capability, the domain name may resolve to a single network address with internal routing to that address from any of the gateway's interfaces. In others, the domain name may resolve to multiple network addresses, one for each interface. In the latter case, if a Call Agent fails to contact the gateway on one of the addresses, it MUST try the alternates.

4.3 Retransmission, and Detection of Lost Associations:

The media gateway control protocol is organized as a set of transactions, each of which is composed of a command and a response, commonly referred to as an acknowledgement. The MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are retransmitted. MGCP entities MUST keep in memory a list of the responses that they sent to recent transactions, i.e., a list of all the responses they sent over the last T-HIST seconds, and a list of the transactions that have not yet finished executing.

The transaction identifiers of incoming commands are compared to the transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction, but simply repeats the response. If a match to a previously responded to transaction is not found, the transaction identifier of the incoming command is compared to the list of transactions that have not yet finished executing. If a match is found, the MGCP entity does not execute the transaction again, but SHOULD simply send a provisional response - a final response will be provided when the execution of the command is complete (see Section 3.5.6 for further detail).

The repetition mechanism is used to guard against four types of possible errors:

* transmission errors, when for example a packet is lost due to noise on a line or congestion in a queue,

* component failure, when for example an interface to a Call Agent becomes unavailable,

* Call Agent failure, when for example an entire Call Agent becomes unavailable,

* failover, when a new Call Agent is "taking over" transparently.

The elements should be able to derive from the past history an estimate of the packet loss rate due to transmission errors. In a properly configured system, this loss rate should be very low, typically less than 1%. If a Call Agent or a gateway has to repeat a message more than a few times, it is very legitimate to assume that something other than a transmission error is occurring. For example, given a loss rate of 1%, the probability that 5 consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a Call Agent that processes 1,000 transactions per second. (Indeed, the number of retransmissions that is considered excessive should be a function of

the prevailing packet loss rate.) We should note that the "suspicion threshold", which we will call "Max1", is normally lower than the "disconnection threshold", which we will call "Max2". Max2 MUST be set to a larger value than Max1.

The MGCP retransmission algorithm is illustrated in the Figure below and explained further in the following:

```
      Command issued: N=0, T=0
              |
              |  +------------ retransmission: N++ <--------------+
              |  |                                                |
              |  |     if T <= T-Max then                         |
              |  |      transmission                              |
              |  |  +-- to new address, <-+<----------------------|--+
              |  |  |       N=0           |                       |  |
              V  V  V                     |                       |  |
          +-----------+                   |                       |  |
      +-->| awaiting  |- new Call Agent ->+   +------------+      |  |
      |   |  response |--- timer elapsed  --->| T > T-Max ?|      |  |
      |   +-----------+                       +------------+      ^  ^
      |          |                             |    |             |  |
      |          v             +-----(yes)-----+   (no)           |  |
      |        (response       |                    |             |  |
      |         received)      |              +------------+      |  |
      |          |             |              | N >= Max1 ?|-(no)>+  |
      |          v             |              +------------+      ^  ^
      |      +--------+        |                    |             |  |
      +<(no)-| final ?|        |                  (yes)           |  |
      ^      +--------+        |                    |             |  |
      |          |             |     (if first address & N=Max1,  |  |
      |          v             |      or last address & N=Max2    |  |
      |        (yes)           |              check DNS)          |  |
      |          |             |                    |             |  |
      |          v             V           +---------------+      |  |
      |        (end)           |           |more addresses?|(yes)-|->+
      |                        |           +---------------+      |
      |                        |                    |             ^
      |                        |                  (no)            |
      |                        |                    |             |
      |                        |              +------------+      |
      |                        |              | N >= Max2 ?|(no)--+
      |                        |              +------------+
      |                        |                    |
      |                        |                  (yes)
      |                        |                    |
      |                        |            +----------------+
      |                        +----------->| T >= 2*T-HIST ?|
      |                                     +----------------+
      |                                       |       |
      |                                     (no)    (yes)
      +---------------<-----------------------+       |
                                                      v
                                                (disconnected)
```

A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after re-transmitting the packet an excessive number of times (typically between 7 and 11 times). In order to account for the possibility of an undetected or in-progress "failover", we modify the classic algorithm as follows:

* We require that the gateway always checks for the presence of a new Call Agent. It can be noticed either by:

  - receiving a command where the NotifiedEntity points to the new Call Agent, or

  - receiving a redirection response pointing to a new Call Agent.

  If a new Call Agent is detected, the gateway MUST start retransmitting outstanding commands for the endpoint(s) redirected to that new Call Agent. Responses to new or old commands are still transmitted to the source address of the command.

* Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than T-MAX. If more than T-MAX time has elapsed, then retransmissions MUST cease. If more than 2*T-HIST has elapsed, then the endpoint becomes disconnected.

* If the number of repetitions for this Call Agent is equal to "Max1", and its domain name was not resolved recently (e.g., within the last 5 seconds or otherwise provisioned), and it is not in the process of being resolved, then the gateway MAY actively query the domain name server in order to detect the possible change of the Call Agent interfaces. Note that the first repetition is the second transmission.

* The gateway may have learned several IP addresses for the call agent. If the number of repetitions for this IP address is greater than or equal to "Max1" and lower than "Max2", and there are more addresses that have not been tried, then the gateway MUST direct the retransmissions to alternate addresses. Also, receipt of explicit network notifications such as, e.g., ICMP network, host, protocol, or port unreachable SHOULD lead the gateway to try alternate addresses (with due consideration to possible security issues).

* If there are no more interfaces to try, and the number of repetitions for this address is Max2, then the gateway SHOULD contact the DNS one more time to see if any other interfaces have become available, unless the domain name was resolved recently (e.g., within the last 5 seconds or otherwise provisioned), or it is already in the process of being resolved. If there still are no more interfaces to try, the gateway is then disconnected and MUST initiate the "disconnected" procedure (see Section 4.4.7).

In order to automatically adapt to network load, MGCP specifies exponentially increasing timers. If the initial timer is set to 200 milliseconds, the loss of a fifth retransmission will be detected after about 6 seconds. This is probably an acceptable waiting delay to detect a failover. The repetitions should continue after that delay not only in order to perhaps overcome a transient connectivity problem, but also in order to allow some more time for the execution of a failover - waiting a total delay of 30 seconds is probably acceptable.

It is however important that the maximum delay of retransmissions be bounded. Prior to any retransmission, it is checked that the time (T) elapsed since the sending of the initial datagram is no greater than T-MAX. If more than T-MAX time has elapsed, retransmissions MUST cease. If more than 2*T-HIST time has elapsed, the endpoint becomes disconnected. The value T-MAX is related to the T-HIST value: the T-HIST value MUST be greater than or equal to T-MAX plus the maximum propagation delay in the network.

The default value for T-MAX is 20 seconds. Thus, if the assumed maximum propagation delay is 10 seconds, then responses to old transactions would have to be kept for a period of at least 30 seconds. The importance of having the sender and receiver agree on these values cannot be overstated.

The default value for Max1 is 5 retransmissions and the default value for Max2 is 7 retransmissions. Both of these values may be altered by the provisioning process.

The provisioning process MUST be able to disable one or both of the Max1 and Max2 DNS queries.

4.4 Race Conditions

MGCP deals with race conditions through the notion of a "quarantine list" and through explicit detection of desynchronization, e.g., for mismatched hook state due to glare for an endpoint.

MGCP does not assume that the transport mechanism will maintain the order of commands and responses. This may cause race conditions, that may be obviated through a proper behavior of the Call Agent. (Note that some race conditions are inherent to distributed systems; they would still occur, even if the commands were transmitted in strict order.)

In some cases, many gateways may decide to restart operation at the same time. This may occur, for example, if an area loses power or transmission capability during an earthquake or an ice storm. When power and transmission are reestablished, many gateways may decide to send "RestartInProgress" commands simultaneously, leading to very unstable operation.

4.4.1 Quarantine List

MGCP controlled gateways will receive "notification requests" that ask them to watch for a list of "events". The protocol elements that determine the handling of these events are the "Requested Events" list, the "Digit Map", the "Quarantine Handling", and the "Detect Events" list.

When the endpoint is initialized, the requested events list only consists of persistent events for the endpoint, and the digit map is assumed empty. At this point, the endpoint MAY use an implicit NotificationRequest with the reserved RequestIdentifier zero ("0") to detect and report a persistent event, e.g., off-hook. A pre-existing off-hook condition MUST here result in the off-hook event being generated as well.

The endpoint awaits the reception of a NotificationRequest command, after which the gateway starts observing the endpoint for occurrences of the events mentioned in the list, including persistent events.

The events are examined as they occur. The action that follows is determined by the "action" parameter associated with the event in the list of requested events, and also by the digit map. The events that are defined as "accumulate" or "accumulate according to digit map" are accumulated in a list of events, the events that are marked as "accumulate according to the digit map" will additionally be accumulated in the "current dial string". This will go on until one event is encountered that triggers a notification which will be sent to the current "notified entity".

The gateway, at this point, will transmit the Notify command and will place the endpoint in a "notification" state. As long as the endpoint is in this notification state, the events that are to be detected on the endpoint are stored in a "quarantine" buffer (FIFO)

for later processing. The events are, in a sense, "quarantined". All events that are specified by the union of the RequestedEvents parameter and the most recently received DetectEvents parameter or, in the absence of the latter, all events that are referred to in the RequestedEvents, SHALL be detected and quarantined, regardless of the action associated with the event. Persistent events are here viewed as implicitly included in RequestedEvents. If the quarantine buffer reaches the capacity of the endpoint, a Quarantine Buffer Overflow event (see Appendix B) SHOULD be generated (when this event is supported, the endpoint MUST ensure it has capacity to include the event in the quarantine buffer). Excess events will now be discarded.

The endpoint exits the "notification state" when the response (whether success or failure) to the Notify command is received. The Notify command may be retransmitted in the "notification state", as specified in Section 3.5 and 4. If the endpoint is or becomes disconnected (see Section 4.3) during this, a response to the Notify command will never be received. The Notify command is then lost and hence no longer considered pending, yet the endpoint is still in the "notification state". Should that occur, completion of the disconnected procedure specified in Section 4.4.7 SHALL then lead the endpoint to exit the "notification state".

When the endpoint exits the "notification state" it resets the list of observed events and the "current dial string" of the endpoint to a null value.

Following that point, the behavior of the gateway depends on the value of the QuarantineHandling parameter in the triggering NotificationRequest command:

If the Call Agent had specified, that it expected at most one notification in response to the notification request command, then the gateway SHALL simply keep on accumulating events in the quarantine buffer until it receives the next notification request command.

If, however, the gateway is authorized to send multiple successive Notify commands, it will proceed as follows. When the gateway exits the "notification state", it resets the list of observed events and the "current dial string" of the endpoint to a null value and starts processing the list of quarantined events, using the already received list of requested events and digit map. When processing these events, the gateway may encounter an event which triggers a Notify command to be sent. If that is the case, the gateway can adopt one of the two following behaviors:

* it can immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included, leaving the unprocessed events in the quarantine buffer,

* or it can attempt to empty the quarantine buffer and transmit a single Notify command reporting several sets of events (in a single list of observed events) and possibly several dial strings. The "current dial string" is reset to a null value after each triggering event. The events that follow the last triggering event are left in the quarantine buffer.

If the gateway transmits a Notify command, the endpoint will reenter and remain in the "notification state" until the acknowledgement is received (as described above). If the gateway does not find a quarantined event that triggers a Notify command, it places the endpoint in a normal state. Events are then processed as they come, in exactly the same way as if a Notification Request command had just been received.

A gateway may receive at any time a new Notification Request command for the endpoint, including the case where the endpoint is disconnected. Activating an embedded Notification Request is here viewed as receiving a new Notification Request as well, except that the current list of ObservedEvents remains unmodified rather than being processed again. When a new notification request is received in the notification state, the gateway SHALL ensure that the pending Notify is received by the Call Agent prior to a new Notify (note that a Notify that was lost due to being disconnected, is no longer considered pending). It does so by using the "piggybacking" functionality of the protocol. The messages will then be sent in a single packet to the current "notified entity". The steps involved are the following:

a) the gateway sends a response to the new notification request.

b) the endpoint is then taken out of the "notification state" without waiting for the acknowledgement of the pending Notify command.

c) a copy of the unacknowledged Notify command is kept until an acknowledgement is received. If a timer elapses, the Notify will be retransmitted.

d) If the gateway has to transmit a new Notify before the previous Notify(s) is acknowledged, it constructs a packet that piggybacks a repetition of the old Notify(s) and the new Notify (ordered by age with the oldest first). This datagram will be sent to the current "notified entity".

f) Gateways that cannot piggyback several messages in the same datagram and hence guarantee in-order delivery of two (or more) Notify's SHALL leave the endpoint in the "notification" state as long as the last Notify is not acknowledged.

The procedure is illustrated by the following diagram:

```
    +-------------------+
    | Processing Events |<--------------------------------------+
    +-------------------+                                       |
             |                                                  |
     Need to send NTFY                                          |
             |                                                  |
             v                                                  |
    +-------------------+                                       |
    | Outstanding NTFY  |---- No -------+                       |
    |                   |               |                       |
    +-------------------+               v                       |
             |                    +-----------+                 |
            Yes                   | Send NTFY |                 |
             |                    +-----------+                 |
             v                          |                       |
    +--------------------+              v                       |
    | Piggyback new NTFY |     +--------------------+           |
    | w. old outstanding |---->| Notification State |           |
    | NTFY(s)            |     +--------------------+           |
    +--------------------+       |               |              |
                             new RQNT        NTFY response      |
                             received        received           |
                                 |               |              |
                                 |               v              |
                                 |        +-------------+       |
                                 |        | Step mode ? |- No ->+
                                 |        +-------------+       ^
                                 |               |              |
                                 |              Yes             |
                                 |               |              |
                                 |               v              |
                                 |       +---------------+      |
                                 |       | Wait for RQNT |      |
                                 |       +---------------+      |
                                 |               |              |
                                 |         RQNT received        |
                                 |               |              |
                                 |               v              |
                                 |       +---------------+      |
                                 +------>| Apply RQNT and|----->+
                                         | send response |
                                         +---------------+
```

Gateways may also attempt to deliver the pending Notify prior to a successful response to the new NotificationRequest by using the "piggybacking" functionality of the protocol. This was in fact required behavior in RFC 2705, however there are several complications in doing this, and the benefits are questionable. In particular, the RFC 2705 mechanism did not guarantee in-order delivery of Notify's and responses to NotificationRequests in general, and hence Call Agents had to handle out-of-order delivery of these messages anyway. The change to optional status is thus backwards compatible while greatly reducing complexity.

After receiving the Notification Request command, the requested events list and digit map (if a new one was provided) are replaced by the newly received parameters, and the current dial string is reset to a null value. Furthermore, when the Notification Request was received in the "notification state", the list of observed events is reset to a null value. The subsequent behavior is conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events (and observed events which in this case is now an empty list), should be discarded, in which case they will be. If the parameter specifies that the quarantined (and observed) events are to be processed, the gateway will start processing the list of quarantined (and observed) events, using the newly received list of requested events and digit map (if provided). When processing these events, the gateway may encounter an event which requires a Notify command to be sent. If that is the case, the gateway will immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included leaving the unprocessed events in the quarantine buffer, and will enter the "notification state".

A new notification request may be received while the gateway has accumulated events according to the previous notification request, but has not yet detected a notification-triggering events, i.e., the endpoint is not in the "notification state". The handling of not-yet-notified events is determined, as with the quarantined events, by the quarantine handling parameter:

* If the quarantine-handling parameter specifies that quarantined events shall be ignored, the observed events list is simply reset.

* If the quarantine-handling parameter specifies that quarantined events shall be processed, the observed event list is transferred to the quarantined event list. The observed event list is then reset, and the quarantined event list is processed.

Call Agents controlling endpoints in lockstep mode SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggybacking mechanism.

4.4.2 Explicit Detection

A key element of the state of several endpoints is the position of the hook. A race condition may occur when the user decides to go off-hook before the Call Agent has the time to ask the gateway to notify an off-hook event (the "glare" condition well known in telephony), or if the user goes on-hook before the Call Agent has the time to request the event's notification.

To avoid this race condition, the gateway MUST check the condition of the endpoint before acknowledging a NotificationRequest. It MUST return an error:

1. If the gateway is requested to notify an "off-hook" transition while the phone is already off-hook, (error code 401 - phone off hook)

2. If the gateway is requested to notify an "on-hook" or "flash hook" condition while the phone is already on-hook (error code 402 - phone on hook).

Additionally, individual signal definitions can specify that a signal will only operate under certain conditions, e.g., ringing may only be possible if the phone is already off-hook. If such prerequisites exist for a given signal, the gateway MUST return the error specified in the signal definition if the prerequisite is not met.

It should be noted, that the condition check is performed at the time the notification request is received, whereas the actual event that caused the current condition may have either been reported, or ignored earlier, or it may currently be quarantined.

The other state variables of the gateway, such as the list of RequestedEvents or list of requested signals, are entirely replaced after each successful NotificationRequest, which prevents any long term discrepancy between the Call Agent and the gateway.

When a NotificationRequest is unsuccessful, whether it is included in a connection-handling command or not, the gateway MUST simply continue as if the command had never been received. As all other transactions, the NotificationRequest MUST operate as an atomic transaction, thus any changes initiated as a result of the command MUST be reverted.

Another race condition may occur when a Notify is issued shortly before the reception by the gateway of a NotificationRequest. The RequestIdentifier is used to correlate Notify commands with NotificationRequest commands thereby enabling the Call Agent to determine if the Notify command was generated before or after the gateway received the new NotificationRequest. This is especially important to avoid deadlocks in "step" mode.

4.4.3 Transactional Semantics

As the potential transaction completion times increase, e.g., due to external resource reservations, a careful definition of the transactional semantics becomes increasingly important. In particular the issue of race conditions, e.g., as it relates to hook-state, must be defined carefully.

An important point to consider is, that the status of a pre-condition (e.g., hook-state) may in fact change between the time a transaction starts and the time it either completes successfully (transaction commit) or fails. In general, we can say that the successful execution of a transaction depends on one or more pre-conditions where the status of one or more of the pre-conditions may change dynamically between the transaction start and transaction commit.

The simplest semantics for this is simply to require that all pre-conditions be met from the time the transaction is initiated until the transaction commits. If any pre-condition is not met before the completion of the transaction, the transaction will also fail.

As an example, consider a transaction that includes a request for the "off-hook" event. When the transaction is initiated the phone is "on-hook" and this pre-condition is therefore met. If the hook-state changes to "off-hook" before the transaction completes, the pre-condition is no longer met, and the transaction therefore immediately fails.

Finally, we need to consider the point in time when a new transaction takes effect and endpoint processing according to an old transaction stops. For example, assume that transaction T1 has been executed successfully and event processing is currently being done according to transaction T1. Now we receive a new transaction T2 specifying new event processing (for example a CreateConnection with an encapsulated NotificationRequest). Since we don't know whether T2 will complete successfully or not, we cannot start processing events according to T2 until the outcome of T2 is known. While we could suspend all event processing until the outcome of T2 is known, this would make for a less responsive system and hence SHOULD NOT be done. Instead, when a new transaction Ty is received and Ty modifies

processing according to an old transaction Tx, processing according to Tx SHOULD remain active for as long as possible, until a successful outcome of Ty is known to occur. If Ty fails, then processing according to Tx will of course continue as usual. Any changes incurred by Ty logically takes effect when Ty commits. Thus, if the endpoint was in the notification state when Ty commits, and Ty contained a NotificationRequest, the endpoint will be taken out of the notification state when Ty commits. Note that this is independent of whether the endpoint was in the notification state when Ty was initiated. For example, a Notify could be generated due to processing according to Tx between the start and commit of Ty. If the commit of Ty leads to the endpoint entering the notification state, a new NotificationRequest (Tz) is needed to exit the notification state. This follows from the fact that transaction execution respects causal order.

Another related issue is the use of wildcards, especially the "all of" wildcard, which may match more than one endpoint. When a command is requested, and the endpoint identifier matches more than one endpoint, transactional semantics still apply. Thus, the command MUST either succeed for all the endpoints, or it MUST fail for all of them. A single response is consequently always issued.

4.4.4 Ordering of Commands, and Treatment of Misorder

MGCP does not mandate that the underlying transport protocol guarantees in-order delivery of commands to a gateway or an endpoint. This property tends to maximize the timeliness of actions, but it has a few drawbacks. For example:

* Notify commands may be delayed and arrive at the Call Agent after the transmission of a new Notification Request command,

* If a new NotificationRequest is transmitted before a previous one is acknowledged, there is no guarantee that the previous one will not be received and executed after the new one.

Call Agents that want to guarantee consistent operation of the endpoints can use the following rules:

1) When a gateway handles several endpoints, commands pertaining to the different endpoints can be sent in parallel, for example following a model where each endpoint is controlled by its own process or its own thread.

2) When several connections are created on the same endpoint, commands pertaining to different connections can be sent in parallel.

3) On a given connection, there should normally be only one outstanding command (create or modify). However, a DeleteConnection command can be issued at any time. In consequence, a gateway may sometimes receive a ModifyConnection command that applies to a previously deleted connection. Such commands will fail, and an error code MUST be returned (error code 515 - incorrect connection-id, is RECOMMENDED).

4) On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter MUST be used to correlate Notify commands with the triggering notification request.

5) In some cases, an implicitly or explicitly wildcarded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The Call Agent should individually delete all connections whose completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wild-carding SHOULD NOT be sent until the wild-carded DeleteConnection command is acknowledged.

6) When commands are embedded within each other, sequencing requirements for all commands must be adhered to. For example a Create Connection command with a Notification Request in it must adhere to the sequencing requirements associated with both CreateConnection and NotificationRequest at the same time.

7) AuditEndpoint and AuditConnection are not subject to any sequencing requirements.

8) RestartInProgress MUST always be the first command sent by an endpoint as defined by the restart procedure. Any other command or non-restart response (see Section 4.4.6), except for responses to auditing, MUST be delivered after this RestartInProgress command (piggybacking allowed).

9) When multiple messages are piggybacked in a single packet, the messages are always processed in order.

10) On a given endpoint, there should normally be only one outstanding EndpointConfiguration command at any time.

Gateways MUST NOT make any assumptions as to whether Call Agents follow these rules or not. Consequently gateways MUST always respond to commands, regardless of whether they adhere to the above rules or not. To ensure consistent operation, gateways SHOULD behave as specified below when one or more of the above rules are not followed:

* Where a single outstanding command is expected (ModifyConnection, NotificationRequest, and EndpointConfiguration), but the same command is received in a new transaction before the old finishes executing, the gateway SHOULD fail the previous command. This includes the case where one or more of the commands were encapsulated. The use of error code 407 (transaction aborted) is RECOMMENDED.

* If a ModifyConnection command is received for a pending CreateConnection command, the ModifyConnection command SHOULD simply be rejected. The use of error code 400 (transient error) is RECOMMENDED. Note that this situation constitutes a Call Agent programming error.

* If a DeleteConnection command is received for a pending CreateConnection or ModifyConnection command, the pending command MUST be aborted. The use of error code 407 (transaction aborted) is RECOMMENDED.

Note, that where reception of a new command leads to aborting an old command, the old command SHOULD be aborted regardless of whether the new command succeeds or not. For example, if a ModifyConnection command is aborted by a DeleteConnection command which itself fails due to an encapsulated NotificationRequest, the ModifyConnection command is still aborted.

4.4.5 Endpoint Service States

As described earlier, endpoints configured for operation may be either in-service or out-of-service. The actual service-state of the endpoint is reflected by the combination of the RestartMethod and RestartDelay parameters, which are sent with RestartInProgress commands (Section 2.3.12) and furthermore may be audited in AuditEndpoint commands (Section 2.3.10).

The service-state of an endpoint affects how it processes a command. An endpoint in-service MUST process any command received, whereas an endpoint that is out-of-service MUST reject non-auditing commands, but SHOULD process auditing commands if possible. For backwards compatibility, auditing commands for an out-of-service endpoint may alternatively be rejected as well. Any command rejected due to an endpoint being out-of-service SHOULD generate error code 501 (endpoint not ready/out-of-service).

Note that (per Section 2.1.2), unless otherwise specified for a command, endpoint names containing the "any of" wildcard only refer to endpoints in-service, whereas endpoint names containing the "all of" wildcard refer to all endpoints, regardless of service state.

The above relationships are illustrated in the table below which shows the current service-states and gateway processing of commands as a function of the RestartInProgress command sent and the response (if any) received to it. The last column also lists (in parentheses) the RestartMethod to be returned if audited:

Restart-Method	Restart-Delay	2xx received ?	Service-State	Response to new command
graceful	zero	Yes/No	In	non-audit: 2xx audit: 2xx (graceful)
graceful	non-zero	Yes/No	In*	non-audit: 2xx audit: 2xx (graceful)
forced	N/A	Yes/No	Out	non-audit: 501 audit: 2xx (forced)
restart	zero	No	In	non-audit: 2xx,405* audit: 2xx (restart)
restart	zero	Yes	In	non-audit: 2xx audit: 2xx (restart)
restart	non-zero	No	Out*	non-audit: 501* audit: 2xx (restart)
restart	non-zero	Yes	Out*	non-audit: 501* audit: 2xx (restart)
discon-nected	zero/non-zero	No	In	non-audit: 2xx, audit: 2xx (disconnected)
discon-nected	zero/non-zero	Yes	In	non-audit: 2xx audit: 2xx (restart)
cancel-graceful	N/A	Yes/No	In	non-audit: 2xx audit: 2xx (restart)

Notes (*):

* The three service-states marked with "*" will change after the expiration of the RestartDelay at which time an updated RestartInProgress command SHOULD be sent.

* If the endpoint returns 2xx when the restart procedure has not yet completed, then in-order delivery MUST still be satisfied, i.e., piggy-backing is to be used. If instead, the command is not processed, 405 SHOULD be returned.

* Following a "restart" RestartInProgress with a non-zero RestartDelay, error code 501 is only returned until the endpoint goes in-service, i.e., until the expiration of the RestartDelay.

4.4.6 Fighting the Restart Avalanche

Let's suppose that a large number of gateways are powered on simultaneously. If they were to all initiate a RestartInProgress transaction, the Call Agent would very likely be swamped, leading to message losses and network congestion during the critical period of service restoration. In order to prevent such avalanches, the following behavior is REQUIRED:

1) When a gateway is powered on, it MUST initiate a restart timer to a random value, uniformly distributed between 0 and a maximum waiting delay (MWD). Care should be taken to avoid synchronicity of the random number generation between multiple gateways that would use the same algorithm.

2) The gateway MUST then wait for either the end of this timer, the reception of a command from the Call Agent, or the detection of a local user activity, such as for example an off-hook transition on a residential gateway.

3) When the timer elapses, when a command is received, or when an activity is detected, the gateway MUST initiate the restart procedure.

The restart procedure simply requires the endpoint to guarantee that the first

* non-audit command, or

* non-restart response (i.e., error codes other than 405, 501, and 520) to a non-audit command

that the Call Agent sees from this endpoint is a "restart" RestartInProgress command. The endpoint is free to take full advantage of piggybacking to achieve this. Endpoints that are considered in-service will have a RestartMethod of "restart", whereas endpoints considered out-of-service will have a RestartMethod of "forced" (also see Section 4.4.5). Commands rejected due to an endpoint not yet having completed the restart procedure SHOULD use error code 405 (endpoint "restarting").

The restart procedure is complete once a success response has been received. If an error response is received, the subsequent behavior depends on the error code in question:

* If the error code indicates a transient error (4xx), then the restart procedure MUST be initiated again (as a new transaction).

* If the error code is 521, then the endpoint is redirected, and the restart procedure MUST be initiated again (as a new transaction). The 521 response MUST have included a NotifiedEntity which then is the "notified entity" towards which the restart is initiated. If it did not include a NotifiedEntity, the response is treated as any other permanent error (see below).

* If the error is any other permanent error (5xx), and the endpoint is not able to rectify the error, then the endpoint no longer initiates the restart procedure on its own (until rebooted/restarted) unless otherwise specified. If a command is received for the endpoint, the endpoint MUST initiate the restart procedure again.

Note that if the RestartInProgress is piggybacked with the response (R) to a command received while restarting, then retransmission of the RestartInProgress does not require piggybacking of the response R. However, while the endpoint is restarting, a resend of the response R does require the RestartInProgress to be piggybacked to ensure in-order delivery of the two.

Should the gateway enter the "disconnected" state while carrying out the restart procedure, the disconnected procedure specified in Section 4.4.7 MUST be carried out, except that a "restart" rather than "disconnected" message is sent during the procedure.

Each endpoint in a gateway will have a provisionable Call Agent, i.e., "notified entity", to direct the initial restart message towards. When the collection of endpoints in a gateway is managed by more than one Call Agent, the above procedure MUST be performed for each collection of endpoints managed by a given Call Agent. The gateway MUST take full advantage of wild-carding to minimize the

number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent. Note that during startup, it is possible for endpoints to start out as being out-of-service, and then become in-service as part of the gateway initialization procedure. A gateway may thus choose to send first a "forced" RestartInProgress for all its endpoints, and subsequently a "restart" RestartInProgress for the endpoints that come in-service. Alternatively, the gateway may simply send "restart" RestartInProgress for only those endpoints that are in-service, and "forced" RestartInProgress for the specific endpoints that are out-of-service. Wild-carding MUST still be used to minimize the number of messages sent though.

The value of MWD is a configuration parameter that depends on the type of the gateway. The following reasoning can be used to determine the value of this delay on residential gateways.

Call agents are typically dimensioned to handle the peak hour traffic load, during which, in average, 10% of the lines will be busy, placing calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 MGCP transactions between each endpoint and the Call Agent. This simple calculation shows that the Call Agent is expected to handle 5 to 6 transactions for each endpoint, every 30 minutes on average, or, to put it otherwise, about one transaction per endpoint every 5 to 6 minutes on average. This suggest that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, residential gateways should adopt a value of 600 seconds for MWD.

The same reasoning suggests that the value of MWD should be much shorter for trunking gateways or for business gateways, because they handle a large number of endpoints, and also because the usage rate of these endpoints is much higher than 10% during the peak busy hour, a typical value being 60%. These endpoints, during the peak hour, are thus expected to contribute about one transaction per minute to the Call Agent load. A reasonable algorithm is to make the value of MWD per "trunk" endpoint six times shorter than the MWD per residential gateway, and also inversely proportional to the number of endpoints that are being restarted. For example MWD should be set to 2.5 seconds for a gateway that handles a T1 line, or to 60 milliseconds for a gateway that handles a T3 line.

4.4.7 Disconnected Endpoints

In addition to the restart procedure, gateways also have a "disconnected" procedure, which MUST be initiated when an endpoint becomes "disconnected" as described in Section 4.3. It should here be noted, that endpoints can only become disconnected when they attempt to communicate with the Call Agent. The following steps MUST be followed by an endpoint that becomes "disconnected":

1. A "disconnected" timer is initialized to a random value, uniformly distributed between 1 and a provisionable "disconnected" initial waiting delay (Tdinit), e.g., 15 seconds. Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways and endpoints that would use the same algorithm.

2. The gateway then waits for either the end of this timer, the reception of a command for the endpoint from the Call Agent, or the detection of a local user activity for the endpoint, such as for example an off-hook transition.

3. When the "disconnected" timer elapses for the endpoint, when a command is received for the endpoint, or when local user activity is detected for the endpoint, the gateway initiates the "disconnected" procedure for the endpoint - if a disconnected procedure was already in progress for the endpoint, it is simply replaced by the new one. Furthermore, in the case of local user activity, a provisionable "disconnected" minimum waiting delay (Tdmin) MUST have elapsed since the endpoint became disconnected or the last time it ended the "disconnected" procedure in order to limit the rate at which the procedure is performed. If Tdmin has not passed, the endpoint simply proceeds to step 2 again, without affecting any disconnected procedure already in progress.

4. If the "disconnected" procedure still left the endpoint disconnected, the "disconnected" timer is then doubled, subject to a provisionable "disconnected" maximum waiting delay (Tdmax), e.g., 600 seconds, and the gateway proceeds with step 2 again (using a new transaction-id).

The "disconnected" procedure is similar to the restart procedure in that it simply states that the endpoint MUST send a RestartInProgress command to the Call Agent informing it that the endpoint was disconnected. Furthermore, the endpoint MUST guarantee that the first non-audit message (non-audit command or response to non-audit command) that the Call Agent sees from this endpoint MUST inform the Call Agent that the endpoint is disconnected (unless the endpoint goes out-of-service). When a command (C) is received, this is achieved by sending a piggy-backed datagram with a "disconnected"

RestartInProgress command and the response to command C to the source address of command C as opposed to the current "notified entity". This piggy-backed RestartInProgress is not automatically retransmitted by the endpoint but simply relies on fate-sharing with the piggy-backed response to guarantee the in-order delivery requirement. The Call Agent still sends a response to the piggy-backed RestartInProgress, however, as usual, the response may be lost. In addition to the piggy-backed RestartInProgress command, a new "disconnected" procedure is triggered by the command received. This will lead to a non piggy-backed copy (i.e., same transaction) of the "disconnected" RestartInProgress command being sent reliably to the current "notified entity".

When the Call Agent learns that the endpoint is disconnected, the Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint. Note that each such "disconnected" procedure will result in a new RestartInProgress command, which will be subject to the normal retransmission procedures specified in Section 4.3. At the end of the procedure, the endpoint may thus still be "disconnected". Should the endpoint go out-of-service while being disconnected, it SHOULD send a "forced" RestartInProgress message as described in Section 2.3.12.

The disconnected procedure is complete once a success response has been received. Error responses are handled similarly to the restart procedure (Section 4.4.6). If the "disconnected" procedure is to be initiated again following an error response, the rate-limiting timer considerations specified above still apply.

Note, that if the RestartInProgress is piggybacked with the response (R) to a command received while being disconnected, then retransmission of this particular RestartInProgress does not require piggybacking of the response R. However, while the endpoint is disconnected, resending the response R does require the RestartInProgress to be piggybacked with the response to ensure the in-order delivery of the two.

If a set of disconnected endpoints have the same "notified entity", and the set of endpoints can be named with a wildcard, the gateway MAY replace the individual disconnected procedures with a suitably wildcarded disconnected procedure instead. In that case, the Restart Delay for the wildcarded "disconnected" RestartInProgress command SHALL be the Restart Delay corresponding to the oldest disconnected procedure replaced. Note that if only a subset of these endpoints subsequently have their "notified entity" changed and/or are no longer disconnected, then that wildcarded disconnected procedure can no longer be used. The remaining individual disconnected procedures MUST then be resumed again.

A disconnected endpoint may wish to send a command (besides RestartInProgress) while it is disconnected. Doing so will only succeed once the Call Agent is reachable again, which raises the question of what to do with such a command meanwhile. At one extreme, the endpoint could drop the command right away, however that would not work very well when the Call Agent was in fact available, but the endpoint had not yet completed the "disconnected" procedure (consider for example the case where a NotificationRequest was just received which immediately resulted in a Notify being generated). To prevent such scenarios, disconnected endpoints SHALL NOT blindly drop new commands to be sent for a period of T-MAX seconds after they receive a non-audit command.

One way of satisfying this requirement is to employ a temporary buffering of commands to be sent, however in doing so, the endpoint MUST ensure, that it:

* does not build up a long queue of commands to be sent,

* does not swamp the Call Agent by rapidly sending too many commands once it is connected again.

Buffering commands for T-MAX seconds and, once the endpoint is connected again, limiting the rate at which buffered commands are sent to one outstanding command per endpoint is considered acceptable (see also Section 4.4.8, especially if using wildcards). If the endpoint is not connected within T-MAX seconds, but a "disconnected" procedure is initiated within T-MAX seconds, the endpoint MAY piggyback the buffered command(s) with that RestartInProgress. Note, that once a command has been sent, regardless of whether it was buffered initially, or piggybacked earlier, retransmission of that command MUST cease T-MAX seconds after the initial send as described in Section 4.3.

This specification purposely does not specify any additional behavior for a disconnected endpoint. Vendors MAY for instance choose to provide silence, play reorder tone, or even enable a downloaded wav file to be played.

The default value for Tdinit is 15 seconds, the default value for Tdmin, is 15 seconds, and the default value for Tdmax is 600 seconds.

4.4.8 Load Control in General

The previous sections have described several MGCP mechanisms to deal with congestion and overload, namely:

* the UDP retransmission strategy which adapts to network and call agent congestion on a per endpoint basis,

* the guidelines on the ordering of commands which limit the number of commands issued in parallel,

* the restart procedure which prevents flooding in case of a restart avalanche, and

* the disconnected procedure which prevents flooding in case of a large number of disconnected endpoints.

It is however still possible for a given set of endpoints, either on the same or different gateways, to issue one or more commands at a given point in time. Although it can be argued, that Call Agents should be sized to handle one message per served endpoint at any given point in time, this may not always be the case in practice. Similarly, gateways may not be able to handle a message for all of its endpoints at any given point in time. In general, such issues can be dealt with through the use of a credit-based mechanism, or by monitoring and automatically adapting to the observed behavior. We opt for the latter approach as follows.

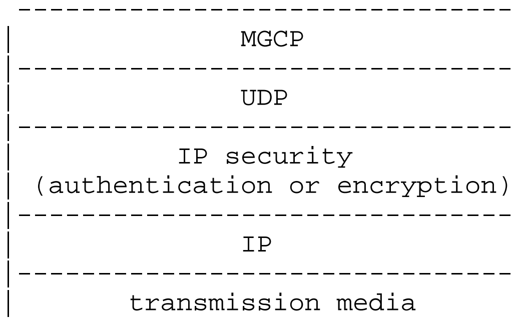
Conceptually, we assume that Call Agents and gateways maintain a queue of incoming transactions to be executed. Associated with this transaction queue is a high-water and a low-water mark. Once the queue length reaches the high-water mark, the entity SHOULD start issuing 101 provisional responses (transaction queued) until the queue length drops to the low-water mark. This applies to new transactions as well as to retransmissions. If the entity is unable to process any new transactions at this time, it SHOULD return error code 409 (processing overload).

Furthermore, gateways SHOULD adjust the sending rate of new commands to a given Call Agent by monitoring the observed response times from that Call Agent to a *set* of endpoints. If the observed smoothed average response time suddenly rises significantly over some threshold, or the gateway receives a 101 (transaction queued) or 409 (overload) response, the gateway SHOULD adjust the sending rate of new commands to that Call Agent accordingly. The details of the smoothing average algorithm, the rate adjustments, and the thresholds involved are for further study, however they MUST be configurable.

Similarly, Call Agents SHOULD adjust the sending rate of new transactions to a given gateway by monitoring the observed response times from that gateway for a *set* of endpoints. If the observed smoothed average response time suddenly rises significantly over some threshold, or the Call Agent receives a 101 (transaction queued) or 409 (overloaded), the Call Agent SHOULD adjust the sending rate of new commands to that gateway accordingly. The details of the smoothing average algorithm, the rate adjustments, and the thresholds involved are for further study, however they MUST be configurable.

5. Security Requirements

Any entity can send a command to an MGCP endpoint. If unauthorized entities could use the MGCP, they would be able to set-up unauthorized calls, or to interfere with authorized calls. We expect that MGCP messages will always be carried over secure Internet connections, as defined in the IP security architecture as defined in RFC 2401, using either the IP Authentication Header, defined in RFC 2402, or the IP Encapsulating Security Payload, defined in RFC 2406. The complete MGCP protocol stack would thus include the following layers:



Adequate protection of the connections will be achieved if the gateways and the Call Agents only accept messages for which IP security provided an authentication service. An encryption service will provide additional protection against eavesdropping, thus preventing third parties from monitoring the connections set up by a given endpoint.

The encryption service will also be requested if the session descriptions are used to carry session keys, as defined in SDP.

These procedures do not necessarily protect against denial of service attacks by misbehaving gateways or misbehaving Call Agents. However, they will provide an identification of these misbehaving entities, which should then be deprived of their authorization through maintenance procedures.

5.1 Protection of Media Connections

MGCP allows Call Agent to provide gateways with "session keys" that can be used to encrypt the audio messages, protecting against eavesdropping.

A specific problem of packet networks is "uncontrolled barge-in". This attack can be performed by directing media packets to the IP address and UDP port used by a connection. If no protection is implemented, the packets will be decoded and the signals will be played on the "line side".

A basic protection against this attack is to only accept packets from known sources, however this tends to conflict with RTP principles. This also has two inconveniences: it slows down connection establishment and it can be fooled by source spoofing:

* To enable the address-based protection, the Call Agent must obtain the source address of the egress gateway and pass it to the ingress gateway. This requires at least one network round trip, and leaves us with a dilemma: either allow the call to proceed without waiting for the round trip to complete, and risk for example "clipping" a remote announcement, or wait for the full round trip and settle for slower call-set-up procedures.

* Source spoofing is only effective if the attacker can obtain valid pairs of source and destination addresses and ports, for example by listening to a fraction of the traffic. To fight source spoofing, one could try to control all access points to the network. But this is in practice very hard to achieve.

An alternative to checking the source address is to encrypt and authenticate the packets, using a secret key that is conveyed during the call set-up procedure. This will not slow down the call set-up, and provides strong protection against address spoofing.

6. Packages

As described in Section 2.1.6, packages are the preferred way of extending MGCP. In this section we describe the requirements associated with defining a package.

A package MUST have a unique package name defined. The package name MUST be registered with the IANA, unless it starts with the characters "x-" or "x+" which are reserved for experimental packages. Please refer to Appendix C for IANA considerations.

A package MUST also have a version defined which is simply a non-negative integer. The default and initial version of a package is zero, the next version is one, etc. New package versions MUST be completely backwards compatible, i.e., a new version of a package MUST NOT redefine or remove any of the extensions provided in an earlier version of the package. If such a need arises, a new package name MUST be used instead.

Packages containing signals of type time-out MAY indicate if the "to" parameter is supported for all the time-out signals in the package as well as the default rounding rules associated with these (see Section 3.2.2.4). If no such definition is provided, each time-out signal SHOULD provide these definitions.

A package defines one or more of the following extensions:

* Actions

* BearerInformation

* ConnectionModes

* ConnectionParameters

* DigitMapLetters

* Events and Signals

* ExtensionParameters

* LocalConnectionOptions

* ReasonCodes

* RestartMethods

* Return codes

For each of the above types of extensions supported by the package, the package definition MUST contain a description of the extension as defined in the following sections. Please note, that package extensions, just like any other extension, MUST adhere to the MGCP grammar.

6.1 Actions

Extension Actions SHALL include:

* The name and encoding of the extension action.

* If the extension action takes any action parameters, then the name, encoding, and possible values of those parameters.

* A description of the operation of the extension action.

* A listing of the actions in this specification the extension can be combined with. If such a listing is not provided, it is assumed that the extension action cannot be combined with any other action in this specification.

* If more than one extension action is defined in the package, then a listing of the actions in the package the extension can be combined with. If such a listing is not provided, it is assumed that the extension action cannot be combined with any other action in the package.

Extension actions defined in two or more different packages SHOULD NOT be used simultaneously, unless very careful consideration to their potential interaction and side-effects has been given.

6.2 BearerInformation

BearerInformation extensions SHALL include:

* The name and encoding of the BearerInformation extension.

* The possible values and encoding of those values that can be assigned to the BearerInformation extension.

* A description of the operation of the BearerInformation extension. As part of this description the default value (if any) if the extension is omitted in an EndpointConfiguration command MUST be defined. It may be necessary to make a distinction between the default value before and after the initial application of the parameter, for example if the parameter retains its previous value once specified, until explicitly altered. If default values are not described, then the extension parameter simply defaults to empty in all EndpointConfiguration commands.

Note that the extension SHALL be included in the result for an AuditEndpoint command auditing the BearerInformation.

6.3 ConnectionModes

Extension Connection Modes SHALL include:

* The name and encoding of the extension connection mode.

* A description of the operation of the extension connection mode.

* A description of the interaction a connection in the extension connection mode will have with other connections in each of the modes defined in this specification. If such a description is not provided, the extension connection mode MUST NOT have any interaction with other connections on the endpoint.

Extension connection modes SHALL NOT be included in the list of modes in a response to an AuditEndpoint for Capabilities, since the package will be reported in the list of packages.

6.4 ConnectionParameters

Extension Connection Parameters SHALL include:

* The name and encoding of the connection parameter extension.

* The possible values and encoding of those values that can be assigned to the connection parameter extension.

* A description of how those values are derived.

Note that the extension connection parameter MUST be included in the result for an AuditConnection command auditing the connection parameters.

6.5 DigitMapLetters

Extension Digit Map Letters SHALL include:

* The name and encoding of the extension digit map letter(s).

* A description of the meaning of the extension digit map letter(s).

Note that extension DigitMapLetters in a digit map do not follow the normal naming conventions for extensions defined in packages. More specifically the package name and slash ("/") will not be part of the extension name, thereby forming a flat and limited name space with potential name clashing.

Therefore, a package SHALL NOT define a digit map letter extension whose encoding has already been used in another package. If two packages have used the same encoding for a digit map letter extension, and those two packages are supported by the same endpoint, the result of using that digit map letter extension is undefined.

Note that although an extension DigitMapLetter does not include the package name prefix and slash ("/") as part of the extension name within a digit map, the package name prefix and slash are included when the event code for the event that matched the DigitMapLetter is reported as an observed event. In other words, the digit map just define the matching rule(s), but the event is still reported like any other event.

6.6 Events and Signals

The event/signal definition SHALL include the precise name of the event/signal (i.e., the code used in MGCP), a plain text definition of the event/signal, and, when appropriate, the precise definition of the corresponding events/signals, for example the exact frequencies of audio signals such as dial tones or DTMF tones.

The package description MUST provide, for each event/signal, the following information:

* The description of the event/signal and its purpose, which SHOULD include the actual signal that is generated by the client (e.g., xx ms FSK tone) as well as the resulting user observed result (e.g., Message Waiting light on/off).

The event code used for the event/signal.

* The detailed characteristics of the event/signal, such as for example frequencies and amplitude of audio signals, modulations and repetitions. Such details may be country specific.

* The typical and maximum duration of the event/signal if applicable.

* If the signal or event can be applied to a connection (across a media stream), it MUST be indicated explicitly. If no such indication is provided, it is assumed that the signal or event cannot be applied to a connection.

For events, the following MUST be provided as well:

* An indication if the event is persistent. By default, events are not persistent - defining events as being persistent is discouraged (see Appendix B for a preferred alternative). Note that persistent

events will automatically trigger a Notify when they occur, unless the Call Agent explicitly instructed the endpoint otherwise. This not only violates the normal MGCP model, but also assumes the Call Agent supports the package in question. Such an assumption is unlikely to hold in general.

* An indication if there is an auditable event-state associated with the event. By default, events do not have auditable event-states.

* If event parameters are supported, it MUST be stated explicitly. The precise syntax and semantics of these MUST then be provided (subject to the grammar provided in Appendix A). It SHOULD also be specified whether these parameters apply to RequestedEvents, ObservedEvents, DetectEvents and EventStates. If not specified otherwise, it is assumed that:

  * they do not apply to RequestedEvents,

  * they do apply to ObservedEvents,

  * they apply in the same way to DetectEvents as they do to RequestedEvents for a given event parameter,

  * they apply in the same way to EventStates as they do to ObservedEvents for a given event parameter.

* If the event is expected to be used in digit map matching, it SHOULD explicitly state so. Note that only events with single letter or digit parameter codes can do this. See Section 2.1.5 for further details.

For signals, the following MUST be provided as well:

* The type of signal (OO, TO, BR).

* Time-Out signals SHOULD have an indication of the default time-out value. In some cases, time-out values may be variable (if dependent on some action to complete such as out-pulsing digits).

* If signal parameters are supported, it MUST be stated explicitly. The precise syntax and semantics of these MUST then be provided (subject to the grammar provided in Appendix A).

* Time-Out signals may also indicate whether the "to" parameter is supported or not as well as what the rounding rules associated with them are. If omitted from the signal definition, the package-wide definition is assumed (see Section 6). If the package definition did not specify this, rounding rules default to the nearest non-

zero second, whereas support for the "to" parameter defaults to "no" for package version zero, and "yes" for package versions one and higher.

The following format is RECOMMENDED for defining events and signals in conformance with the above:

Symbol	Definition	R	S Duration

where:

* Symbol indicates the event code used for the event/signal, e.g., "hd".

* Definition gives a brief definition of the event/signal

* R contains an "x" if the event can be detected or one or more of the following symbols:

  - "P" if the event is persistent.

  - "S" if the events is an event-state that may be audited.

  - "C" if the event can be detected on a connection.

* S contains one of the following if it is a signal:

  - "OO" if the signal is On/Off signal.

  - "TO" if the signal is a Time-Out signal.

  - "BR" if the signal is a Brief signal.

* S also contains:

  - "C" if the signal can be applied on a connection.

The table SHOULD then be followed by a more comprehensive description of each event/signal defined.

6.6.1 Default and Reserved Events

All packages that contain Time-Out type signals contain the operation failure ("of") and operation complete ("oc") events, irrespective of whether they are provided as part of the package description or not. These events are needed to support Time-Out signals and cannot be overridden in packages with Time-Out signals. They MAY be extended if necessary, however such practice is discouraged.

If a package without Time-Out signals does contain definitions for the "oc" and "of" events, the event definitions provided in the package MAY over-ride those indicated here. Such practice is however discouraged and is purely allowed to avoid potential backwards compatibility problems.

It is considered good practice to explicitly mention that the two events are supported in accordance with their default definitions, which are as follows:

Symbol	Definition	R	S	Duration
oc	Operation Complete	x		
of	Operation Failure	x		

Operation complete (oc): The operation complete event is generated when the gateway was asked to apply one or several signals of type TO on the endpoint or connection, and one or more of those signals completed without being stopped by the detection of a requested event such as off-hook transition or dialed digit. The completion report should carry as a parameter the name of the signal that came to the end of its live time, as in:

```
O: G/oc(G/rt)
```

In this case, the observed event occurred because the "rt" signal in the "G" package timed out.

If the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: G/oc(G/rt@0A3F58)
```

When the operation complete event is requested, it cannot be parameterized with any event parameters. When the package name is omitted (which is discouraged) as part of the signal name, the default package is assumed.

Operation failure (of): The operation failure event is generated when the endpoint was asked to apply one or several signals of type TO on the endpoint or connection, and one or more of those signals failed prior to timing out. The completion report should carry as a parameter the name of the signal that failed, as in:

```
O: G/of(G/rt)
```

In this case a failure occurred in producing the "rt" signal in the "G" package.

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: G/of(G/rt@0A3F58)
```

When the operation failure event is requested, event parameters can not be specified. When the package name is omitted (which is discouraged), the default package name is assumed.

6.7 ExtensionParameters

Extension parameter extensions SHALL include:

* The name and encoding of the extension parameter.

* The possible values and encoding of those values that can be assigned to the extension parameter.

* For each of the commands defined in this specification, whether the extension parameter is Mandatory, Optional, or Forbidden in requests as well as responses. Note that extension parameters SHOULD NOT normally be mandatory.

* A description of the operation of the extension parameter. As part of this description the default value (if any) if the extension is omitted in a command MUST be defined. It may be necessary to make a distinction between the default value before and after the initial application of the parameter, for example if the parameter retains its previous value once specified, until explicitly altered. If default values are not described, then the extension parameter simply defaults to empty in all commands.

* Whether the extension can be audited in AuditEndpoint and/or AuditConnection as well as the values returned. If nothing is specified, then auditing of the extension parameter can only be done for AuditEndpoint, and the value returned SHALL be the current value for the extension. Note that this may be empty.

6.8 LocalConnectionOptions

LocalConnectionOptions extensions SHALL include:

* The name and encoding of the LocalConnectionOptions extension.

* The possible values and encoding of those values that can be assigned to the LocalConnectionOptions extension.

* A description of the operation of the LocalConnectionOptions extension. As part of this description the following MUST be specified:

  - The default value (if any) if the extension is omitted in a CreateConnection command.

  - The default value if omitted in a ModifyConnection command. This may be to simply retain the previous value (if any) or to apply the default value. If nothing is specified, the current value is retained if possible.

  - If Auditing of capabilities will result in the extension being returned, then a description to that effect as well as with what possible values and their encoding (note that the package itself will always be returned). If nothing is specified, the extension SHALL NOT be returned when auditing capabilities.

Also note, that the extension MUST be included in the result for an AuditConnection command auditing the LocalConnectionOptions.

6.9 Reason Codes

Extension reason codes SHALL include:

* The number for the reason code. The number MUST be in the range 800 to 899.

* A description of the extension reason code including the circumstances that leads to the generation of the reason code. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension reason code correctly.

Note that the extension reason code may have to be provided in the result for an AuditEndpoint command auditing the reason code.

6.10 RestartMethods

Extension Restart Methods SHALL include:

* The name and encoding for the restart method.

* A description of the restart method including the circumstances that leads to the generation of the restart method. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension restart method correctly.

* An indication of whether the RestartDelay parameter is to be used with the extension. If nothing is specified, it is assumed that it is not to be used. In that case, RestartDelay MUST be ignored if present.

* If the restart method defines a service state, the description MUST explicitly state and describe this. In that case, the extension restart method can then be provided in the result for an AuditEndpoint command auditing the restart method.

6.11 Return Codes

Extension Return Codes SHALL include:

* The number for the extension return code. The number MUST be in the range 800 to 899.

* A description of the extension return code including the circumstances that leads to the generation of the extension return code. Those circumstances SHOULD be limited to events caused by another extension defined in the package to ensure the recipient will be able to interpret the extension return code correctly.

7. Versions and Compatibility

7.1 Changes from RFC 2705

RFC 2705 was issued in October 1999, as the last update of draft version 0.5. This updated document benefits from further implementation experience. The main changes from RFC 2705 are:

* Contains several clarifications, editorial changes and resolution of known inconsistencies.

* Firmed up specification language in accordance with RFC 2119 and added RFC 2119 conventions section.

* Clarified behavior of mixed wild-carding in endpoint names.

* Deleted naming requirement about having first term identify the physical gateway when the gateway consists of multiple physical gateways. Also added recommendations on wild-carding naming usage from the right only, as well as mixed wildcard usage.

* Clarified that synonymous forms and values for endpoint names are not freely interchangeable.

* Allowed IPv6 addresses in endpoint names.

* Clarified Digit Map matching rules.

* Added missing semantics for symbols used in digit maps.

* Added Timer T description in Digit Maps.

* Added recommendation to support digit map sizes of at least 2048 bytes per endpoint.

* Clarified use of wildcards in several commands.

* Event and Signal Parameters formally defined for events and signals.

* Persistent events now allowed in base MGCP protocol.

* Added additional detail on connection wildcards.

* Clarified behavior of loopback, and continuity test connection modes for mixing and multiple connections in those modes.

* Modified BearerInformation to be conditional optional in the EndpointConfiguration command.

* Clarified "swap audio" action operation for one specific scenario and noted that operation for other scenarios is undefined.

* Added recommendation that all implementations support PCMU encoding for interoperability.

* Changed Bandwidth LocalConnectionOptions value from excluding to including overhead from the IP layer and up for consistency with SDP.

* Clarified that mode of second connection in a CreateConnection command will be set to "send/receive".

* Type of service default changed to zero.

* Additional detail on echo cancellation, silence suppression, and gain control. Also added recommendation for Call Agents not to specify handling of echo cancellation and gain control.

* Added requirement for a connection to have a RemoteConnectionDescriptor in order to use the "network loopback" and "network continuity test" modes.

* Removed procedures and specification for NAS's (will be provided as package instead).

* Removed procedures and specification for ATM (will be provided as package instead).

* Added missing optional NotifiedEntity parameter to the DeleteConnection (from the Call Agent) MGCI command.

* Added optional new MaxMGCPDatagram RequestedInfo code for AuditEndpoint to enable auditing of maximum size of MGCP datagrams supported.

* Added optional new PackageList RequestedInfo code for AuditEndpoint to enable auditing of packages with a package version number. PackageList parameter also allowed with return code 518 (unsupported package).

* Added missing attributes in Capabilities.

* Clarified that at the expiration of a non-zero restart delay, an updated RestartInProgress should be sent. Also clarified that a new NotifiedEntity can only be returned in response to a RestartInProgress command.

* Added Response Acknowledgement response (return code 000) and included in three-way handshake.

* ResponseAck parameter changed to be allowed in all commands.

* Added return codes 101, 405, 406, 407, 409, 410, 503, 504, 505, 506, 507, 508, 509, 533, 534, 535, 536, 537, 538, 539, 540, 541, and defined return codes in range 800-899 to be package specific return codes. Additional text provided for some return codes and additional detail on how to handle unknown return codes added.

* Added reason code 903, 904, 905 and defined reason codes 800-899 to be package specific reason codes.

* Added section clarifying codec negotiation procedure.

* Clarified that resource reservation parameters in a ModifyConnection command defaults to the current value used.

* Clarified that connection mode is optional in ModifyConnection commands.

* Corrected LocalConnectionDescriptor to be optional in response to CreateConnection commands (in case of failure).

* Clarified that quoted-strings are UTF-8 encoded and interchangeability of quoted strings and unquoted strings.

* Clarified that Transaction Identifiers are compared as numerical values.

* Clarified bit-ordering for Type Of Service LocalConnectionOptions.

* Clarified the use of RequestIdentifier zero.

* Added example sections for commands, responses, and some call flows.

* Corrected usage of and requirements for SDP to be strictly RFC 2327 compliant.

* Added requirement that all MGCP implementations must support MGCP datagrams up to at least 4000 bytes. Also added new section on Maximum Datagram Size, Fragmentation and reassembly.

* Generalized piggybacking retransmission scheme to only state underlying requirements to be satisfied.

* Clarified the section on computing retransmission timers.

* Clarified operation of long-running transactions, including provisional responses, retransmissions and failures.

* Enhanced description of provisional responses and interaction with three-way handshake.

* Enhanced description of fail-over and the role of "notified entity". An empty "notified entity" has been allowed, although strongly discouraged.

* Clarified retransmission procedure and removed "wrong key" considerations from it. Also fixed inconsistencies between Max1 and Max2 retransmission boundaries and the associated flow diagram.

* Updated domain name resolution for retransmission procedure to incur less overhead when multiple endpoints are retransmitting.

* Removed requirement for in-order delivery of NotificationRequests response and Notify commands. Notify commands are still delivered in-order.

* Clarified that activating an embedded Notification Request does not clear the list of ObservedEvents.

* Defined interactions between disconnected state and notification state.

* Added section on transactional semantics.

* Defined gateway behavior when multiple interacting transactions are received.

* Additional details provided on service states. Clarified relationship between endpoint service states, restart methods, and associated processing of commands.

* Clarified operation for transitioning from "restart procedure" to "disconnected state".

* Allowed auditing commands and responses to bypass the "restart" and "disconnected" procedures.

* Clarified operation of "disconnected procedure" and in particular the operation of piggy-backed "disconnected" RestartInProgress messages.

* Added option to aggregate "disconnected" RestartInProgress messages under certain conditions to reduce message volume.

* Defined additional behavior for endpoints wishing to send commands while in the "disconnected" state.

* Added new section on Load Control in General which includes two new error codes (101 and 409) to handle overload.

* Deleted the "Proposed MoveConnection command".

* Removed packages from protocol specification (will be provided in separate documents instead).

* Package concept formally extended to be primary extension mechanism now allowing extensions for the following to be defined in packages as well:

  - BearerInformation

  - LocalConnectionOptions

  - ExtensionParameters

  - Connection Modes

  - Actions

  - Digit Map Letters

  - Connection Parameters

  - Restart Methods

  - Reason Codes

  - Return Codes

* Requirements and suggested format for package definitions added.

* Defined "operation complete" and "operation failure" events to be automatically present in packages with Time-Out signals.

* Deleted list of differences that were prior to RFC 2705.

* Added Base Package to deal with quarantine buffer overflow, ObservedEvents overflow, embedded NotificationRequest failure, and to enable events to be requested persistently. A new "Message" command is included as well.

* IANA registration procedures for packages and other extensions added.

* Updated grammar to fix known errors and support new extensions in a backwards compatible manner. Added new (optional) PackageList and MaxMGCPDatagram for auditing. Changed explicit white space rules in some productions to make grammar more consistent.

* Connection Mode interaction table added.

* Added additional detail on virtual endpoint naming conventions. Also added suggested gateway endpoint convention and a "Range Wildcard" option to the Endpoint Naming Conventions.

8. Security Considerations

Security issues are discussed in section 5.

9. Acknowledgements

Special thanks are due to the authors of the original MGCP 1.0 specification: Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema, and Scott Picket.

We also want to thank the many reviewers who provided advice on the design of SGCP and then MGCP, notably Sankar Ardhanari, Francois Berard, David Auerbach, Bob Biskner, David Bukovinsky, Charles Eckel, Mario Edini, Ed Guy, Barry Hoffner, Jerry Kamitses, Oren Kudevitzki, Rajesh Kumar, Troy Morley, Dave Oran, Jeff Orwick, John Pickens, Lou Rubin, Chip Sharp, Paul Sijben, Kurt Steinbrenner, Joe Stone, and Stuart Wray.

The version 0.1 of MGCP was heavily inspired by the "Internet Protocol Device Control" (IPDC) designed by the Technical Advisory Committee set up by Level 3 Communications. Whole sets of text were retrieved from the IP Connection Control protocol, IP Media Control protocol, and IP Device Management. The authors wish to acknowledge the contribution to these protocols made by Ilya Akramovich, Bob Bell, Dan Brendes, Peter Chung, John Clark, Russ Dehlinger, Andrew Dugan, Isaac Elliott, Cary FitzGerald, Jan Gronski, Tom Hess, Geoff Jordan, Tony Lam, Shawn Lewis, Dave Mazik, Alan Mikhak, Pete O'Connell, Scott Pickett, Shyamal Prasad, Eric Presworsky, Paul Richards, Dale Skran, Louise Spergel, David Sprague, Raj Srinivasan, Tom Taylor and Michael Thomas.

10. References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[4] Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

[5] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[6] Handley, M., Perkins, C. and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[7] Rosenberg, J., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schulzrinne, H. and E. Schooler, "Session Initiation Protocol (SIP)", RFC 3261, June 2002.

[8] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[9] ITU-T, Recommendation Q.761, "FUNCTIONAL DESCRIPTION OF THE ISDN USER PART OF SIGNALING SYSTEM No. 7", (Malaga-Torremolinos, 1984; modified at Helsinki, 1993).

[10] ITU-T, Recommendation Q.762, "GENERAL FUNCTION OF MESSAGES AND SIGNALS OF THE ISDN USER PART OF SIGNALING SYSTEM No. 7", (MalagaTorremolinos, 1984; modified at Helsinki, 1993).

[11] ITU-T, Recommendation H.323 (02/98), "PACKET-BASED MULTIMEDIA COMMUNICATIONS SYSTEMS".

[12] ITU-T, Recommendation H.225, "Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems".

[13] ITU-T, Recommendation H.245 (02/98), "CONTROL PROTOCOL FOR MULTIMEDIA COMMUNICATION".

[14] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[15] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[16] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[17] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[18] Stevens, W. Richard, "TCP/IP Illustrated, Volume 1, The Protocols", Addison-Wesley, 1994.

[19] Allman, M., Paxson, V. "On Estimating End-to-End Network Path Properties", Proc. SIGCOMM'99, 1999.

[20] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[21] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989.

[22] Bellcore, "LSSGR: Switching System Generic Requirements for Call Control Using the Integrated Services Digital Network User Part (ISDNUP)", GR-317-CORE, Issue 2, December 1997.

[23] Narten, T., and Alvestrand H., "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, October 1998.

Appendix A: Formal Syntax Description of the Protocol

In this section, we provide a formal description of the protocol syntax, following the "Augmented BNF for Syntax Specifications" defined in RFC 2234. The syntax makes use of the core rules defined in RFC 2234, Section 6.1, which are not included here. Furthermore, the syntax follows the case-sensitivity rules of RFC 2234, i.e., MGCP is case-insensitive (but SDP is not). It should be noted, that ABNF does not provide for implicit specification of linear white space and MGCP messages MUST thus follow the explicit linear white space rules provided in the grammar below. However, in line with general robustness principles, implementers are strongly encouraged to tolerate additional linear white space in messages received.

```
MGCPMessage = MGCPCommand / MGCPResponse

MGCPCommand = MGCPCommandLine 0*(MGCPParameter) [EOL *SDPinformation]

MGCPCommandLine = MGCPVerb 1*(WSP) transaction-id 1*(WSP)
                        endpointName 1*(WSP) MGCPversion EOL

MGCPVerb = "EPCF" / "CRCX" / "MDCX" / "DLCX" / "RQNT"
         / "NTFY" / "AUEP" / "AUCX" / "RSIP" / extensionVerb

extensionVerb  = ALPHA 3(ALPHA / DIGIT) ; experimental starts with X

transaction-id = 1*9(DIGIT)

endpointName      = LocalEndpointName "@" DomainName
LocalEndpointName = LocalNamePart 0*("/" LocalNamePart)
LocalNamePart     = AnyName / AllName / NameString
AnyName           = "$"
AllName           = "*"
NameString        = 1*(range-of-allowed-characters)
; VCHAR except "$", "*", "/", "@"
range-of-allowed-characters  = %x21-23 / %x25-29 / %x2B-2E
                             / %x30-3F / %x41-7E

DomainName = 1*255(ALPHA / DIGIT / "." / "-")    ; as defined
           / "#" number                          ; in RFC 821
           / "[" IPv4address / IPv6address "]"   ; see RFC 2373

; Rewritten to ABNF from RFC 821
number =  1*DIGIT

;From RFC 2373
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
```

```
; this production, while occurring in RFC2373, is not referenced
; IPv6prefix  = hexpart "/" 1*2DIGIT
hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq  = hex4 *( ":" hex4)
hex4    = 1*4HEXDIG

MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
                             [1*(WSP) ProfileName]
ProfileName = VCHAR *( WSP / VCHAR)

MGCPParameter = ParameterValue EOL

; Check infoCode if more parameter values defined
; Most optional values can only be omitted when auditing
ParameterValue = ("K"  ":" 0*(WSP)  [ResponseAck])
               / ("B"  ":" 0*(WSP)  [BearerInformation])
               / ("C"  ":" 0*(WSP)  CallId)
               / ("I"  ":" 0*(WSP)  [ConnectionId])
               / ("N"  ":" 0*(WSP)  [NotifiedEntity])
               / ("X"  ":" 0*(WSP)  [RequestIdentifier])
               / ("L"  ":" 0*(WSP)  [LocalConnectionOptions])
               / ("M"  ":" 0*(WSP)  ConnectionMode)
               / ("R"  ":" 0*(WSP)  [RequestedEvents])
               / ("S"  ":" 0*(WSP)  [SignalRequests])
               / ("D"  ":" 0*(WSP)  [DigitMap])
               / ("O"  ":" 0*(WSP)  [ObservedEvents])
               / ("P"  ":" 0*(WSP)  [ConnectionParameters])
               / ("E"  ":" 0*(WSP)  ReasonCode)
               / ("Z"  ":" 0*(WSP)  [SpecificEndpointID])
               / ("Z2" ":" 0*(WSP)  SecondEndpointID)
               / ("I2" ":" 0*(WSP)  SecondConnectionID)
               / ("F"  ":" 0*(WSP)  [RequestedInfo])
               / ("Q"  ":" 0*(WSP)  QuarantineHandling)
               / ("T"  ":" 0*(WSP)  [DetectEvents])
               / ("RM" ":" 0*(WSP)  RestartMethod)
               / ("RD" ":" 0*(WSP)  RestartDelay)
               / ("A"  ":" 0*(WSP)  [Capabilities])
               / ("ES" ":" 0*(WSP)  [EventStates])
               / ("PL" ":" 0*(WSP)  [PackageList])    ; Auditing only
               / ("MD" ":" 0*(WSP)  MaxMGCPDatagram)  ; Auditing only
               / (extensionParameter ":" 0*(WSP) [parameterString])

; A final response may include an empty ResponseAck
ResponseAck =  confirmedTransactionIdRange
               *( ","  0*(WSP) confirmedTransactionIdRange )

confirmedTransactionIdRange = transaction-id ["-" transaction-id]
```

```
BearerInformation = BearerAttribute 0*("," 0*(WSP) BearerAttribute)
BearerAttribute   = ("e" ":" BearerEncoding)
                  / (BearerExtensionName [":" BearerExtensionValue])
BearerExtensionName  = PackageLCOExtensionName
BearerExtensionValue = LocalOptionExtensionValue
BearerEncoding = "A" / "mu"

CallId = 1*32(HEXDIG)

; The audit request response may include a list of identifiers
ConnectionId = 1*32(HEXDIG) 0*("," 0*(WSP) 1*32(HEXDIG))
SecondConnectionID = ConnectionId

NotifiedEntity = [LocalName "@"] DomainName [":" portNumber]
LocalName  = LocalEndpointName            ; No internal structure

portNumber = 1*5(DIGIT)

RequestIdentifier = 1*32(HEXDIG)

LocalConnectionOptions = LocalOptionValue 0*(WSP)
                           0*("," 0*(WSP) LocalOptionValue 0*(WSP))
LocalOptionValue = ("p"  ":" packetizationPeriod)
                 / ("a"  ":" compressionAlgorithm)
                 / ("b"  ":" bandwidth)
                 / ("e"  ":" echoCancellation)
                 / ("gc" ":" gainControl)
                 / ("s"  ":" silenceSuppression)
                 / ("t"  ":" typeOfService)
                 / ("r"  ":" resourceReservation)
                 / ("k"  ":" encryptiondata)
                 / ("nt" ":" ( typeOfNetwork /
                                    supportedTypeOfNetwork))
                 / (LocalOptionExtensionName
                         [":" LocalOptionExtensionValue])

Capabilities    =  CapabilityValue 0*(WSP)
                     0*("," 0*(WSP) CapabilityValue 0*(WSP))
CapabilityValue = LocalOptionValue
                / ("v" ":" supportedPackages)
                / ("m" ":" supportedModes)

PackageList     = pkgNameAndVers 0*("," pkgNameAndVers)
pkgNameAndVers  = packageName ":" packageVersion
packageVersion  = 1*(DIGIT)

packetizationPeriod  = 1*4(DIGIT) ["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
```

```
algorithmName         = 1*(SuitableLCOCharacter)
bandwidth             = 1*4(DIGIT) ["-" 1*4(DIGIT)]
echoCancellation      = "on" / "off"
gainControl           = "auto" / ["-"] 1*4(DIGIT)
silenceSuppression    = "on" / "off"
typeOfService         = 1*2(HEXDIG)     ; 1 hex only for capabilities
resourceReservation   = "g" / "cl" / "be"

;encryption parameters are coded as in SDP (RFC 2327)
;NOTE: encryption key may contain an algorithm as specified in RFC 1890
encryptiondata = ( "clear" ":" encryptionKey )
               / ( "base64" ":" encodedEncryptionKey )
               / ( "uri" ":" URItoObtainKey )
               / ( "prompt" ) ; defined in SDP, not usable in MGCP!

encryptionKey = 1*(SuitableLCOCharacter) / quotedString
; See RFC 2045
encodedEncryptionKey = 1*(ALPHA / DIGIT / "+" / "/" / "=")
URItoObtainKey = 1*(SuitableLCOCharacter) / quotedString

typeOfNetwork = "IN" / "ATM" / "LOCAL" / OtherTypeOfNetwork
; Registered with IANA - see RFC 2327
OtherTypeOfNetwork     = 1*(SuitableLCOCharacter)
supportedTypeOfNetwork = typeOfNetwork *(";" typeOfNetwork)

supportedModes    = ConnectionMode 0*(";" ConnectionMode)

supportedPackages = packageName 0*(";" packageName)

packageName = 1*(ALPHA / DIGIT / HYPHEN) ; Hyphen neither first or last

LocalOptionExtensionName = VendorLCOExtensionName
                         / PackageLCOExtensionName
                         / OtherLCOExtensionName
VendorLCOExtensionName   = "x" ("+"/"-") 1*32(SuitableExtLCOCharacter)
PackageLCOExtensionName  = packageName "/"
                           1*32(SuitablePkgExtLCOCharacter)
; must not start with "x-" or "x+"
OtherLCOExtensionName    = 1*32(SuitableExtLCOCharacter)

LocalOptionExtensionValue = (1*(SuitableExtLCOValChar)
                                                    / quotedString)
                             *(";" (1*(SuitableExtLCOValChar)
                                                      / quotedString))

;Note: No "data" mode.
ConnectionMode = "sendonly" / "recvonly" / "sendrecv"
               / "confrnce" / "inactive" / "loopback"
```

```
               / "conttest" / "netwloop" / "netwtest"
               / ExtensionConnectionMode
ExtensionConnectionMode = PkgExtConnectionMode
PkgExtConnectionMode    = packageName "/" 1*(ALPHA / DIGIT)

RequestedEvents = requestedEvent 0*("," 0*(WSP) requestedEvent)
requestedEvent  = (eventName ["(" requestedActions ")"])
                / (eventName "(" requestedActions ")"
                                       "(" eventParameters ")" )
eventName = [(packageName / "*") "/"]
                (eventId / "all" / eventRange
                                        / "*" / "#") ; for DTMF
                              ["@" (ConnectionId / "$" / "*")]
eventId = 1*(ALPHA / DIGIT / HYPHEN)   ; Hyphen neither first nor last
eventRange = "[" 1*(DigitMapLetter / (DIGIT "-" DIGIT) /
                        (DTMFLetter "-" DTMFLetter)) "]"
DTMFLetter = "A" / "B" / "C" / "D"

requestedActions = requestedAction 0*("," 0*(WSP) requestedAction)
requestedAction  = "N" / "A" / "D" / "S" / "I" / "K"
                 / "E" "(" EmbeddedRequest ")"
                 / ExtensionAction
ExtensionAction  = PackageExtAction
PackageExtAction = packageName "/" Action ["(" ActionParameters ")"]
Action           = 1*ALPHA
ActionParameters = eventParameters        ; May contain actions

;NOTE: Should tolerate different order when receiving, e.g., for NCS.
EmbeddedRequest = (      "R" "(" EmbeddedRequestList ")"
                    ["," 0*(WSP) "S" "(" EmbeddedSignalRequest ")"]
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"]      )
                / (      "S" "(" EmbeddedSignalRequest ")"
                    ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"] )
                / (      "D" "(" EmbeddedDigitMap ")" )

EmbeddedRequestList   = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap

SignalRequests   = SignalRequest 0*("," 0*(WSP) SignalRequest )
SignalRequest    = eventName [ "(" eventParameters ")" ]

eventParameters  = eventParameter 0*("," 0*(WSP) eventParameter)
eventParameter   = eventParameterValue
                 / eventParameterName "=" eventParameter
                 / eventParameterName "(" eventParameters ")"
eventParameterString = 1*(SuitableEventParamCharacter)
eventParameterName   = eventParameterString
```

```
eventParameterValue  = eventParameterString / quotedString

DigitMap           = DigitString  / "(" DigitStringList ")"
DigitStringList    = DigitString 0*( "|" DigitString )
DigitString        = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition      = DigitMapLetter / DigitMapRange
; NOTE "X" is now included
DigitMapLetter     = DIGIT / "#" / "*" / "A" / "B" / "C" / "D" / "T"
                   / "X" / ExtensionDigitMapLetter
ExtensionDigitMapLetter = "E" / "F" / "G" / "H" / "I" / "J" / "K"
                        / "L" / "M" / "N" / "O" / "P" / "Q" / "R"
                        / "S" / "U" / "V" / "W" / "Y" / "Z"
; NOTE "[x]" is now allowed
DigitMapRange = "[" 1*DigitLetter "]"
DigitLetter   = *((DIGIT "-" DIGIT) / DigitMapLetter)

ObservedEvents = SignalRequests

EventStates    = SignalRequests

ConnectionParameters = ConnectionParameter
                        0*( "," 0*(WSP) ConnectionParameter )

ConnectionParameter  = ( "PS" "=" packetsSent )
                     / ( "OS" "=" octetsSent )
                     / ( "PR" "=" packetsReceived )
                     / ( "OR" "=" octetsReceived )
                     / ( "PL" "=" packetsLost )
                     / ( "JI" "=" jitter )
                     / ( "LA" "=" averageLatency )
                     / ( ConnectionParameterExtensionName
                           "=" ConnectionParameterExtensionValue )
packetsSent     = 1*9(DIGIT)
octetsSent      = 1*9(DIGIT)
packetsReceived = 1*9(DIGIT)
octetsReceived  = 1*9(DIGIT)
packetsLost     = 1*9(DIGIT)
jitter          = 1*9(DIGIT)
averageLatency  = 1*9(DIGIT)

ConnectionParameterExtensionName = VendorCPExtensionName
                                 /    PackageCPExtensionName
VendorCPExtensionName  = "X" "-" 2*ALPHA
PackageCPExtensionName = packageName "/" CPName
CPName = 1*(ALPHA / DIGIT / HYPHEN)
ConnectionParameterExtensionValue = 1*9(DIGIT)
```

```
MaxMGCPDatagram = 1*9(DIGIT)

ReasonCode = 3DIGIT
             [1*(WSP) "/" packageName]     ; Only for 8xx
             [WSP 1*(%x20-7E)]

SpecificEndpointID = endpointName
SecondEndpointID   = endpointName

RequestedInfo = infoCode 0*("," 0*(WSP) infoCode)

infoCode = "B" / "C" / "I" / "N" / "X" / "L" / "M" / "R" / "S"
         / "D" / "O" / "P" / "E" / "Z" / "Q" / "T" / "RC" / "LC"
         / "A" / "ES" / "RM" / "RD" / "PL" / "MD" / extensionParameter

QuarantineHandling = loopControl / processControl
                   / (loopControl "," 0*(WSP) processControl )
loopControl    = "step" / "loop"
processControl = "process" / "discard"

DetectEvents = SignalRequests

RestartMethod = "graceful" / "forced" / "restart" / "disconnected"
              / "cancel-graceful" / extensionRestartMethod
extensionRestartMethod = PackageExtensionRM
PackageExtensionRM     = packageName "/" 1*32(ALPHA / DIGIT / HYPHEN)
RestartDelay = 1*6(DIGIT)

extensionParameter = VendorExtensionParameter
                   / PackageExtensionParameter
                   / OtherExtensionParameter
VendorExtensionParameter  = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
PackageExtensionParameter = packageName "/"
                            1*32(ALPHA / DIGIT / HYPHEN)
; must not start with "x-" or x+"
OtherExtensionParameter   = 1*32(ALPHA / DIGIT / HYPHEN)

;If first character is a double-quote, then it is a quoted-string
parameterString = (%x21 / %x23-7F) *(%x20-7F) ; first and last must not
                                              ; be white space
                    / quotedString

MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                                        *2(EOL *SDPinformation)

MGCPResponseLine = responseCode 1*(WSP) transaction-id
                        [1*(WSP) "/" packageName]    ; Only for 8xx
                             [WSP responseString] EOL
```

```
responseCode = 3DIGIT
responseString = *(%x20-7E)

SuitablePkgExtLCOCharacter =  SuitableLCOCharacter

SuitableExtLCOCharacter = DIGIT / ALPHA / "+" / "-" / "_" / "&"
              / "!" / "'" / "|" / "=" / "#" / "?"
              / "." / "$" / "*" /       "@" / "[" / "]"
              / "^" / "`" / "{" / "}" / "~"

SuitableLCOCharacter   = SuitableExtLCOCharacter / "/"

SuitableExtLCOValChar  = SuitableLCOCharacter / ":"

; VCHAR except """, "(", ")", ",", and "="
SuitableEventParamCharacter = %x21 / %x23-27 / %x2A-2B
                            / %x2D-3C / %x3E-7E

; NOTE: UTF8 encoded
quotedString  = DQUOTE 0*(quoteEscape / quoteChar) DQUOTE
quoteEscape   = DQUOTE DQUOTE
quoteChar = (%x00-21 / %x23-FF)

EOL = CRLF / LF

HYPHEN = "-"

; See RFC 2327 for proper SDP grammar instead.
SDPinformation = SDPLine CRLF *(SDPLine CRLF)        ; see RFC 2327
SDPLine        = 1*(%x01-09 / %x0B / %x0C / %x0E-FF) ; for proper def.
```

Appendix B: Base Package

Package name: B
Version: 0

The MGCP specification defines a base package which contains a set of events and extension parameters that are of general use to the protocol. Although not required, it is highly RECOMMENDED to support this package as it provides important functionality for the base protocol.

B.1 Events

The table below lists the events:

Symbol	Definition	R	S Duration
enf(##)	embedded RQNT failure	x	
oef	observed events full	x	
qbo	quarantine buffer overflow	x	

The events are defined as follows:

Embedded NotificationRequest failure (enf):
The Embedded NotificationRequest Failure (enf) event is generated when an embedded Notification Request failure occurs. When the event is requested, it should be as part of the Embedded NotificationRequest itself. When the event is reported, it may be parameterized with an error code (see Section 2.4) detailing the error that occurred. When requested, it cannot be parameterized.

Observed events full (oef):
The event is generated when the endpoint is unable to accumulate any more events in the list of ObservedEvents. If this event occurs, and it is not used to trigger a Notify, subsequent events that should have been added to the list will be lost.

Quarantine buffer overflow (qbo):
The event is generated when the quarantine buffer overflows and one or more events have been lost.

B.2 Extension Parameters

B.2.1 PersistentEvents

PersistentEvents: A list of events that the gateway is requested to detect and report persistently. The parameter is optional but can be provided in any command where the DetectEvents parameter can be provided. The initial default value of the parameter is empty. When the parameter is omitted from a command, it retains its current value. When the parameter is provided, it completely replaces the current value. Providing an event in this list, is similar (but preferable) to defining that particular event as being persistent. The current list of PersistentEvents will implicitly apply to the current as well as subsequent NotificationRequests, however no glare detection etc. will be performed (similarly to DetectEvents). If an event provided in this list is included in a RequestedEvents list, the action and event parameters used in the RequestedEvents will replace the action and event parameters associated with the event in the PersistentEvents list for the life of the RequestedEvents list, after which the PersistentEvents action and event parameters are restored. Events with event states requested through this parameter will be included in the list of EventStates if audited.

PersistentEvents can also be used to detect events on connections. Use of the "all connections" wildcard is straightforward, whereas using PersistentEvents with one or more specific connections must be considered carefully. Once the connection in question is deleted, a subsequent NotificationRequest without a new PersistentEvents value will fail (error code 515 - incorrect connection-id, is RECOMMENDED), as it implicitly refers to the deleted connection.

The parameter generates the relevant error codes from the base protocol, e.g., error code 512 if an unknown event is specified.

The PersistentEvents parameter can be audited, in which case it will return its current value. Auditing of RequestedEvents is not affected by this extension, i.e., events specified in this list are not automatically reported when auditing RequestedEvents.

The parameter name for PersistentEvents is "PR" and it is defined by the production:

```
PersistentEvents = "PR" ":" 0*WSP  [RequestedEvents]
```

The following example illustrates the use of the parameter:

```
B/PR: L/hd(N), L/hf(N), L/hu(N), B/enf, B/oef, B/qbo
```

which instructs the endpoint to persistently detect and report off-hook, hook-flash, and on-hook. It also instructs the endpoint to persistently detect and report Embedded Notification Request failure, Observed events full, and Quarantine buffer overflow.

B.2.2 NotificationState

NotificationState is a RequestedInfo parameter that can be audited with the AuditEndpoint command. It can be used to determine if the endpoint is in the notification state or not.

The parameter is forbidden in any command. In responses, it is a valid response parameter for AuditEndpoint only.

It is defined by the following grammar:

```
NotificationState        = "NS" ":" 0*WSP NotificationStateValue
NotificationStateValue   = "ns" / "ls" / "o"
```

It is requested as part of auditing by including the parameter code in RequestedInfo, as in:

```
F: B/NS
```

The response parameter will contain the value "ns" if the endpoint is in the "notification state", the value "ls" if the endpoint is in the "lockstep state" (i.e., waiting for an RQNT after a response to a NTFY has been received when operating in "step" mode), or the value "o" otherwise, as for example:

```
B/NS: ns
```

B.3 Verbs

MGCP packages are not intended to define new commands, however an exception is made in this case in order to add an important general capability currently missing, namely the ability for the gateway to send a generic message to the Call Agent.

The definition of the new command is:

```
ReturnCode
<-- Message(EndpointId
               [, ...])
```

EndpointId is the name for the endpoint(s) in the gateway which is issuing the Message command. The identifier MUST be a fully qualified endpoint identifier, including the domain name of the gateway. The local part of the endpoint name MUST NOT use the "any of" wildcard.

The only parameter specified in the definition of the Message command is the EndpointId, however, it is envisioned that extensions will define additional parameters to be used with the Message command. Such extensions MUST NOT alter or otherwise interfere with the normal operation of the basic MGCP protocol. They may however define additional capabilities above and beyond that provided by the basic MGCP protocol. For example, an extension to enable the gateway to audit the packages supported by the Call Agent could be defined, whereas using the Message command as an alternative way of reporting observed events would be illegal, as that would alter the normal MGCP protocol behavior.

In order to not interfere with normal MGCP operation, lack of a response to the Message command MUST NOT lead the endpoint to become disconnected. The endpoint(s) MUST be prepared to handle this transparently and continue normal processing unaffected.

If the endpoint(s) receive a response indicating that the Call Agent does not support the Message command, the endpoint(s) MUST NOT send a Message command again until the current "notified entity" has changed. Similarly, if the endpoint(s) receive a response indicating that the Call Agent does not support one or more parameters in the Message command, the endpoint(s) MUST NOT send a Message command with those parameters again until the current "notified entity" has changed.

The Message command is encoded as MESG, as shown in the following example:

```
MESG 1200 aaln/1@rgw.whatever.net MGCP 1.0
```

Appendix C: IANA Considerations

C.1 New MGCP Package Sub-Registry

The IANA has established a new sub-registry for MGCP packages under http://www.iana.org/assignments/mgcp-packages.

Packages can be registered with the IANA according to the following procedure:

The package MUST have a unique string name which MUST NOT start with the two characters "x-" or "x+".

The package title, name, and version (zero assumed by default) MUST be registered with IANA as well as a reference to the document that describes the package. The document MUST have a stable URL and MUST be contained on a public web server.

Packages may define one or more Extension Digit Map Letters, however these are taken from a limited and flat name space. To prevent name clashing, IANA SHALL NOT register a package that defines an Extension Digit Map Letter already defined in another package registered by IANA. To ease this task, such packages SHALL contain the line "Extension Digit Map Letters: " followed by a list of the Extension Digit Map Letters defined in the package at the beginning of the package definition.

A contact name, e-mail and postal address for the package MUST be provided. The contact information SHALL be updated by the defining organization as necessary.

Finally, prior to registering a package, the IANA MUST have a designated expert [23] review the package. The expert reviewer will send e-mail to the IANA on the overall review determination.

C.2 New MGCP Package

This document defines a new MGCP Base Package in Appendix B, which has been registered by IANA.

C.3 New MGCP LocalConnectionOptions Sub-Registry

The IANA has established a new sub-registry for MGCP LocalConnectionOptions under http://www.iana.org/assignments/mgcp-localconnectionoptions.

Packages are the preferred extension mechanism, however for backwards compatibility, local connection options beyond those provided in this specification can be registered with IANA. Each such local connection option MUST have a unique string name which MUST NOT start with "x-" or "x+". The local connection option field name and encoding name MUST be registered with IANA as well as a reference to the document that describes the local connection option. The document MUST have a stable URL and MUST be contained on a public web server.

A contact name, e-mail and postal address for the local connection option MUST be provided. The contact information SHALL be updated by the defining organization as necessary.

Finally, prior to registering a LocalConnectionOption, the IANA MUST have a designated expert [23] review the LocalConnectionOption. The expert reviewer will send e-mail to the IANA on the overall review determination.

Appendix D: Mode Interactions

An MGCP endpoint can establish one or more media streams. These streams are either incoming (from a remote endpoint) or outgoing (generated at the handset microphone). The "connection mode" parameter establishes the direction and generation of these streams. When there is only one connection to an endpoint, the mapping of these streams is straightforward; the handset plays the incoming stream over the handset speaker and generates the outgoing stream from the handset microphone signal, depending on the mode parameter.

However, when several connections are established to an endpoint, there can be many incoming and outgoing streams. Depending on the connection mode used, these streams may interact differently with each other and the streams going to/from the handset.

The table below describes how different connections SHALL be mixed when one or more connections are concurrently "active". An active connection is here defined as a connection that is in one of the following modes:

* "send/receive"
* "send only"
* "receive only"
* "conference"

Connections in "network loopback", "network continuity test", or "inactive" modes are not affected by connections in the "active" modes. The Table uses the following conventions:

* Ai is the incoming media stream from Connection A
* Bi is the incoming media stream from Connection B
* Hi is the incoming media stream from the Handset Microphone
* Ao is the outgoing media stream to Connection A
* Bo is the outgoing media stream to Connection B
* Ho is the outgoing media stream to the Handset earpiece
* NA indicates no stream whatsoever (assuming there are no signals applied on the connection)

"netw" in the following table indicates either "netwloop" or "netwtest" mode.

Connection B Mode	Connection A Mode					
	sendonly	recvonly	sendrecv	confrnce	inactive	netw
Send only	Ao=Hi Bo=Hi Ho=NA	Ao=NA Bo=Hi Ho=Ai	Ao=Hi Bo=Hi Ho=Ai	Ao=Hi Bo=Hi Ho=Ai	Ao=NA Bo=Hi Ho=NA	Ao=Ai Bo=Hi Ho=NA
recv only		Ao=NA Bo=NA Ho=Ai+Bi	Ao=Hi Bo=NA Ho=Ai+Bi	Ao=Hi Bo=NA Ho=Ai+Bi	Ao=NA Bo=NA Ho=Bi	Ao=Ai Bo=NA Ho=Bi
send recv			Ao=Hi Bo=Hi Ho=Ai+Bi	Ao=Hi Bo=Hi Ho=Ai+Bi	Ao=NA Bo=Hi Ho=Bi	Ao=Ai Bo=Hi Ho=Bi
conf rnce				Ao=Hi+Bi Bo=Hi+Ai Ho=Ai+Bi	Ao=NA Bo=Hi Ho=Bi	Ao=Ai Bo=Hi Ho=Bi
Inac tive					Ao=NA Bo=NA Ho=NA	Ao=Ai Bo=NA Ho=NA
netw						Ao=Ai Bo=Bi Ho=NA

If there are three or more "active" connections they will still interact as defined in the table above with the outgoing media streams mixed for each interaction (union of all streams). If internal resources are used up and the streams cannot be mixed, the gateway MUST return an error (error code 403 or 502, not enough resources, are RECOMMENDED).

Appendix E: Endpoint Naming Conventions

The following sections provide some RECOMMENDED endpoint naming conventions.

E.1 Analog Access Line Endpoints

The string "aaln", should be used as the first term in a local endpoint name for analog access line endpoints. Terms following "aaln" should follow the physical hierarchy of the gateway so that if the gateway has a number of RJ11 ports, the local endpoint name could look like the following:

```
aaln/#
```

where "#" is the number of the analog line (RJ11 port) on the gateway.

On the other hand, the gateway may have a number of physical plug-in units, each of which contain some number of RJ11 ports, in which case, the local endpoint name might look like the following:

```
aaln/<unit #>/#
```

where <unit #> is the number of the plug in unit in the gateway and "#" is the number of the analog line (RJ11 port) on that unit. Leading zeroes MUST NOT be used in any of the numbers ("#") above.

E.2 Digital Trunks

The string "ds" should be used for the first term of digital endpoints with a naming convention that follows the physical and digital hierarchy such as:

```
ds/<unit-type1>-<unit #>/<unit-type2>-<unit #>/.../<channel #>
```

where: <unit-type> identifies the particular hierarchy level. Some example values of <unit-type> are: "s", "su", "oc3", "ds3", "e3", "ds2", "e2", "ds1", "e1" where "s" indicates a slot number and "su" indicates a sub-unit within a slot. Leading zeroes MUST NOT be used in any of the numbers ("#") above.

The <unit #> is a decimal number which is used to reference a particular instance of a <unit-type> at that level of the hierarchy. The number of levels and naming of those levels is based on the physical hierarchy within the media gateway.

E.3 Virtual Endpoints

Another type of endpoint is one that is not associated with a physical interface (such as an analog or digital endpoint). This type of endpoint is called a virtual endpoint and is often used to represent some DSP resources that gives the endpoint some capability. Examples are announcement, IVR or conference bridge devices. These devices may have multiple instances of DSP functions so that a possible naming convention is:

```
<virtual-endpoint-type>/<endpoint-#>
```

where <virtual-endpoint-type> may be some string representing the type of endpoint (such as "ann" for announcement server or "cnf" for conference server) and <endpoint-#> would identify a particular virtual endpoint within the device. Leading zeroes MUST NOT be used in the number ("#") above. If the physical hierarchy of the server includes plug-in DSP cards, another level of hierarchy in the local endpoint name may be used to describe the plug in unit.

A virtual endpoint may be created as the result of using the "any of" wildcard. Similarly, a virtual endpoint may cease to exist once the last connection on the virtual endpoint is deleted. The definition of the virtual endpoint MUST detail both of these aspects.

When a <virtual-endpoint-type> creates and deletes virtual endpoints automatically, there will be cases where no virtual endpoints exist at the time a RestartInProgress command is to be issued. In such cases, the gateway SHOULD simply use the "all of" wildcard in lieu of any specific <endpoint-#> as in, e.g.:

```
ann/*@mygateway.whatever.net
```

If the RestartInProgress command refers to all endpoints in the gateway (virtual or not), the <virtual-endpoint-id> can be omitted as in, e.g.:

```
*@mygateway.whatever.net
```

Commands received by the gateway will still have to refer to an actual endpoint (possibly created by that command by use of the "any of" wildcard) in order for the command to be processed though.

E.4 Media Gateway

MGCP only defines operation on endpoints in a media gateway. It may be beneficial to define an endpoint that represents the gateway itself as opposed to the endpoints managed by the gateway. Implementations that wish to do so should use the local endpoint name "mg" (for media gateway) as in:

```
mg@mygateway.whatever.net
```

Note that defining such an endpoint does not change any of the protocol semantics, i.e., the "mg" endpoint and other endpoints (e.g., digital trunks) in the gateway are still independent endpoints and MUST be treated as such. For example, RestartInProgress commands MUST still be issued for all endpoints in the gateway as usual.

E.5 Range Wildcards

As described in Section 2.1.2, the MGCP endpoint naming scheme defines the "all of" and "any of" wildcards for the individual terms in a local endpoint name. While the "all of" wildcard is very useful for reducing the number of messages, it can by definition only be used when we wish to refer to all instances of a given term in the local endpoint name. Furthermore, in the case where a command is to be sent by the gateway to the Call Agent, the "all of" wildcard can only be used if all of the endpoints named by it have the same "notified entity". Implementations that prefer a finer-grained wildcarding scheme can use the range wildcarding scheme described here.

A range wildcard is defined as follows:

```
RangeWildcard    = "[" NumericalRange *( "," NumericalRange ) "]"
NumericalRange   = 1*(DIGIT) [ "-" 1*(DIGIT) ]
```

Note that white space is not permitted. Also, since range wildcards use the character "[" to indicate the start of a range, the "[" character MUST NOT be used in endpoint names that use range wildcards. The length of a range wildcard SHOULD be bounded to a reasonably small value, e.g., 128 characters.

Range wildcards can be used anywhere an "all of" wildcard can be used. The semantics are identical for the endpoints named. However, it MUST be noted, that use of the range wildcarding scheme requires support on both the gateway and the Call Agent. Therefore, a gateway MUST NOT assume that it's Call Agent supports range wildcarding and vice versa. In practice, this typically means that both the gateway and Call Agent will need to be provisioned consistently in order to

use range wildcards. Also, if a gateway or Call Agent using range wildcards receives an error response that could indicate a possible endpoint naming problem, they MUST be able to automatically revert to not using range wildcards.

The following examples illustrates the use of range wildcards:

```
ds/ds1-1/[1-12]
ds/ds1-1/[1,3,20-24]
ds/ds1-[1-2]/*
ds/ds3-1/[1-96]
```

The following example illustrates how to use it in a command:

```
RSIP 1204 ds/ds3-1/[1-96]@tgw-18.whatever.net MGCP 1.0
RM: restart
RD: 0
```

Appendix F: Example Command Encodings

This appendix provides examples of commands and responses shown with the actual encoding used. Examples are provided for each command. All commentary shown in the commands and responses is optional.

F.1 NotificationRequest

The first example illustrates a NotificationRequest that will ring a phone and look for an off-hook event:

```
RQNT 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: l/hd(N)
S: l/rg
```

The response indicates that the transaction was successful:

```
200 1201 OK
```

The second example illustrates a NotificationRequest that will look for and accumulate an off-hook event, and then provide dial-tone and accumulate digits according to the digit map provided. The "notified entity" is set to "ca@ca1.whatever.net:5678", and since the SignalRequests parameter is empty (it could have been omitted as well), all currently active TO signals will be stopped. All events in the quarantine buffer will be processed, and the list of events to detect in the "notification" state will include fax tones in addition to the "requested events" and persistent events:

```
RQNT 1202 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: L/hd(A, E(S(L/dl),R(L/oc, L/hu, D/[0-9#*T](D))))
D: (0T|00T|#xxxxxxx|*xx|91xxxxxxxxxx|9011x.T)
S:
Q: process
T: G/ft
```

The response indicates that the transaction was successful:

```
200 1202 OK
```

F.2 Notify

The example below illustrates a Notify message that notifies an off-hook event followed by a 12-digit number beginning with "91". A transaction identifier correlating the Notify with the NotificationRequest it results from is included. The command is sent to the current "notified entity", which typically will be the actual value supplied in the NotifiedEntity parameter, i.e., "ca@ca1.whatever.net:5678" - a failover situation could have changed this:

```
NTFY 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: L/hd,D/9,D/1,D/2,D/0,D/1,D/8,D/2,D/9,D/4,D/2,D/6,D/6
```

The Notify response indicates that the transaction was successful:

```
200 2002 OK
```

F.3 CreateConnection

The first example illustrates a CreateConnection command to create a connection on the endpoint specified. The connection will be part of the specified CallId. The LocalConnectionOptions specify that G.711 mu-law will be the codec used and the packetization period will be 10 ms. The connection mode will be "receive only":

```
CRCX 1204 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
```

The response indicates that the transaction was successful, and a connection identifier for the newly created connection is therefore included. A session description for the new connection is included as well - note that it is preceded by an empty line.

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The second example illustrates a CreateConnection command containing a notification request and a RemoteConnectionDescriptor:

```
CRCX 1205 aaln/1@rgw-2569.whatever.net MGCP 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789AD
R: L/hd
S: L/rg

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction failed, because the phone was already off-hook. Consequently, neither a connection-id nor a session description is returned:

```
401 1205 Phone off-hook
```

Our third example illustrates the use of the provisional response and the three-way handshake. We create another connection and acknowledge the previous response received by using the response acknowledgement parameter:

```
CRCX 1206 aaln/1@rgw-2569.whatever.net MGCP 1.0
K: 1205
C: A3C47F21456789F0
L: p:10, a:PCMU
M: inactive

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
```

A provisional response is returned initially:

```
100 1206 Pending
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

A little later, the final response is received:

```
200 1206 OK
K:
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The Call Agent acknowledges the final response as requested:

```
000 1206
```

and the transaction is complete.

F.4 ModifyConnection

The first example shows a ModifyConnection command that simply sets the connection mode of a connection to "send/receive" - the "notified entity" is set as well:

```
MDCX 1209 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
N: ca@ca1.whatever.net
M: sendrecv
```

The response indicates that the transaction was successful:

```
200 1209 OK
```

In the second example, we pass a session description and include a notification request with the ModifyConnection command. The endpoint will start playing ring-back tones to the user:

```
MDCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: L/hu
S: G/rt

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
```

The response indicates that the transaction was successful:

```
200 1206 OK
```

F.5 DeleteConnection (from the Call Agent)

In this example, the Call Agent simply instructs the gateway to delete the connection "FDE234C8" on the endpoint specified:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The response indicates success, and that the connection was deleted. Connection parameters for the connection are therefore included as well:

```
250 1210 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

F.6 DeleteConnection (from the gateway)

In this example, the gateway sends a DeleteConnection command to the Call Agent to instruct it that a connection on the specified endpoint has been deleted. The ReasonCode specifies the reason for the deletion, and Connection Parameters for the connection are provided as well:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
E: 900 - Hardware error
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48
```

The Call Agent sends a success response to the gateway:

```
200 1210 OK
```

F.7 DeleteConnection (multiple connections from the Call Agent)

In the first example, the Call Agent instructs the gateway to delete all connections related to call "A3C47F21456789F0" on the specified endpoint:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0
C: A3C47F21456789F0
```

The response indicates success and that the connection(s) were deleted:

```
250 1210 OK
```

In the second example, the Call Agent instructs the gateway to delete all connections related to all of the endpoints specified:

```
DLCX 1210 aaln/*@rgw-2567.whatever.net MGCP 1.0
```

The response indicates success:

```
250 1210 OK
```

F.8 AuditEndpoint

In the first example, the Call Agent wants to learn what endpoints are present on the gateway specified, hence the use of the "all of" wild-card for the local portion of the endpoint-name:

```
AUEP 1200 *@rgw-2567.whatever.net MGCP 1.0
```

The gateway indicates success and includes a list of endpoint names:

```
200 1200 OK
Z: aaln/1@rgw-2567.whatever.net
Z: aaln/2@rgw-2567.whatever.net
```

In the second example, the capabilities of one of the endpoints is requested:

```
AUEP 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0
F: A
```

The response indicates success and the capabilities as well. Two codecs are supported, however with different capabilities. Consequently two separate capability sets are returned:

```
200 1201 OK
A: a:PCMU, p:10-100, e:on, s:off, v:L;S, m:sendonly;
         recvonly;sendrecv;inactive;netwloop;netwtest
A: a:G729, p:30-90, e:on, s:on, v:L;S, m:sendonly;
         recvonly;sendrecv;inactive;confrnce;netwloop
```

Note that the carriage return in the Capabilities lines are shown for formatting reasons only - they are not permissible in a real implementation.

In the third example, the Call Agent audits several types of information for the endpoint:

```
AUEP 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0
F: R,D,S,X,N,I,T,O,ES
```

The response indicates success:

```
200 2002 OK
R: L/hu,L/oc(N),D/[0-9](N)
D:
S: L/vmwi(+)
X: 0123456789B1
N: [128.96.41.12]
I: 32F345E2
T: G/ft
O: L/hd,D/9,D/1,D/2
ES: L/hd
```

The list of requested events contains three events. Where no package name is specified, the default package is assumed. The same goes for actions, so the default action - Notify - must therefore be assumed for the "L/hu" event. The omission of a value for the "digit map" means the endpoint currently does not have a digit map. There are currently no active time-out signals, however the OO signal "vmwi" is currently on and is consequently included - in this case it was parameterized, however the parameter could have been excluded. The current "notified entity" refers to an IP-address and only a single connection exists for the endpoint. The current value of DetectEvents is "G/ft", and the list of ObservedEvents contains the four events specified. Finally, the event-states audited reveals that the phone was off-hook at the time the transaction was processed.

F.9 AuditConnection

The first example shows an AuditConnection command where we audit the CallId, NotifiedEntity, LocalConnectionOptions, Connection Mode, LocalConnectionDescriptor, and the Connection Parameters:

```
AUCX 2003 aaln/1@rgw-2567.whatever.net MGCP 1.0
I: 32F345E2
F: C,N,L,M,LC,P
```

The response indicates success and includes information for the RequestedInfo:

```
200 2003 OK
C: A3C47F21456789F0
N: ca@ca1.whatever.net
L: p:10, a:PCMU
M: sendrecv
P: PS=395, OS=22850, PR=615, OR=30937, PL=7, JI=26, LA=47

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0
```

In the second example, we request to audit RemoteConnectionDescriptor and LocalConnectionDescriptor:

```
AUCX 1203 aaln/2@rgw-2567.whatever.net MGCP 1.0
I: FDE234C8
F: RC,LC
```

The response indicates success, and includes information for the RequestedInfo. In this case, no RemoteConnectionDescriptor exists, hence only the protocol version field is included for the RemoteConnectionDescriptor:

```
200 1203 OK

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0

v=0
```

F.10 RestartInProgress

The first example illustrates a RestartInProgress message sent by an gateway to inform the Call Agent that the specified endpoint will be taken out-of-service in 300 seconds:

```
RSIP 1200 aaln/1@rgw-2567.whatever.net MGCP 1.0
RM: graceful
RD: 300
```

The Call Agent’s response indicates that the transaction was successful:

```
200 1200 OK
```

In the second example, the RestartInProgress message sent by the gateway informs the Call Agent, that all of the gateway’s endpoints are being placed in-service in 0 seconds, i.e., they are currently in service. The restart delay could have been omitted as well:

```
RSIP 1204 *@rgw-2567.whatever.net MGCP 1.0
RM: restart
RD: 0
```

The Call Agent’s response indicates success, and furthermore provides the endpoints in question with a new "notified entity":

```
200 1204 OK
N: CA-1@whatever.net
```

Alternatively, the command could have failed with a new "notified entity" as in:

```
521 1204 OK
N: CA-1@whatever.net
```

In that case, the command would then have to be retried in order to satisfy the "restart procedure", this time going to Call Agent "CA-1@whatever.net".

Appendix G: Example Call Flows

The message flow tables in this section use the following abbreviations:

* rgw = Residential Gateway

* ca = Call Agent

* n+ = step ’n’ is repeated one or more times

Note that any use of upper and lower case within the text of the messages is to aid readability and is not in any way a requirement. The only requirement involving case is to be case insensitive at all times.

G.1 Restart

G.1.1 Residential Gateway Restart

The following table shows a message sequence that might occur when a call agent (ca) is contacted by two independent residential gateways (rgw1 and rgw2) which have restarted.

Table F.1: Residential Gateway Restart

step#	usr1	rgw1	ca	rgw2	usr2
1		rsip ->			
			<- ack		
2			<- auep		
		ack ->			
3+			<- rqnt		
		ack ->			
4				<- rsip	
			ack ->		
5			auep ->		
				<- ack	
6+			rqnt ->		
				<- ack	

Step 1 - RestartInProgress (rsip) from rgw1 to ca

rgw1 uses DNS to determine the domain name of ca and send to the default port of 2727. The command consists of the following:

```
rsip 1 *@rgw1.whatever.net mgcp 1.0
rm: restart
```

The "*" is used to inform ca that all endpoints of rgw1 are being restarted, and "restart" is specified as the restart method. The Call Agent "ca" acknowledges the command with an acknowledgement message containing the transaction-id (in this case 1) for the command. It sends the acknowledgement to rgw1 using the same port specified as the source port for the rsip. If none was indicated, it uses the default port of 2727.

```
200 1 ok
```

A response code is mandatory. In this case, "200", indicates "the requested transaction was executed normally". The response string is optional. In this case, "ok" is included as an additional description.

Step 2 - AuditEndpoint (auep) from ca to rgw1

The command consists of the following:

```
auep 153 *@rgw1.whatever.net mgcp 1.0
```

The "*" is used to request audit information from rgw1 of all its endpoints. rgw1 acknowledges the command with an acknowledgement message containing the transaction-id (in this case 153) of the command, and it includes a list of its endpoints. In this example, rgw1 has two endpoints, aaln/1 and aaln/2.

```
200 153 ok
Z: aaln/1@rgw1.whatever.net
Z: aaln/2@rgw1.whatever.net
```

Once it has the list of endpoint ids, ca may send individual AuditEndpoint commands in which the "*" is replaced by the id of the given endpoint. As its response, rgw1 would replace the endpoint id list returned in the example with the info requested for the endpoint. This optional message exchange is not shown in this example.

Step 3 - NotificationRequest (rqnt) from ca to each endpoint of rgw1

In this case, ca sends two rqnts, one for aaln/1:

```
rqnt 154 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a0
```

and a second for aaln/2:

```
rqnt 155 aaln/2@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a1
```

Note that in the requested events parameter line, the event is fully specified as "l/hd", i.e., with the package name, in order to avoid any potential ambiguity. This is the recommended behavior. For the sake of clarity, the action, which in this case is to Notify, is explicitly specified by including the "(n)". If no action is specified, Notify is assumed as the default regardless of the event. If any other action is desired, it must be stated explicitly.

The expected response from rgw1 to these requests is an acknowledgement from aaln/1 as follows:

```
200 154 ok
```

and from aaln/2:

```
200 155 ok
```

Step 4 RestartInProgress (rsip) from rgw2 to ca

```
rsip 0 *@rgw2.whatever.net mgcp 1.0
rm: restart
```

followed by the acknowledgement from ca:

```
200 0 ok
```

Step 5 - AuditEndpoint (auep) from ca to rgw2

```
auep 156 *@rgw2.whatever.net mgcp 1.0
```

followed by an acknowledgement from rgw2:

```
200 156 ok
z: aaln/1@rgw2.whatever.net
z: aaln/2@rgw2.whatever.net
```

Step 6 - NotificationRequest (rqnt) from ca to each endpoint of rgw2

```
rqnt 157 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a2
```

followed by:

```
rqnt 158 aaln/2@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 3456789a3
```

with rgw2 acknowledging for aaln/1:

```
200 157 ok
```

and for aaln/2:

```
200 158 ok
```

G.1.2 Call Agent Restart

The following table shows the message sequence which occurs when a call agent (ca) restarts. How it determines the address information of the gateways, in this case rgw1 and rgw2, is not covered in this document. For interoperability, it is RECOMMENDED to provide the ability to configure the call agent to send AUEP (*) to specific addresses and ports.

Table F.2: Residential Gateway Restart

#	usr1	rgw1	ca	rgw2	usr2
1			<- auep		
		ack ->			
2+			<- rqnt		
		ack ->			
3			auep ->		
				<- ack	
4+			rqnt ->		
				<- ack	

Step 1 - AuditEndpoint (auep) from ca to rgw1

The command consists of the following:

```
auep 0 *@rgw1.whatever.net mgcp 1.0
```

The "*" is used to request audit information from rgw1 of all its endpoints. rgw1 acknowledges the command with an acknowledgement message containing the transaction id (in this case 0) of the command, and it includes a list of its endpoints. In this example, rgw1 has two endpoints, aaln/1 and aaln/2.

```
200 0 ok
z: aaln/1@rgw1.whatever.net
z: aaln/2@rgw1.whatever.net
```

Once it has the list of endpoint ids, ca may send individual AuditEndpoint commands in which the "*" is replaced by the id of the given endpoint. As its response, rgw1 would replace the endpoint id list returned in the example with the info requested for the endpoint. This optional message exchange is not shown in this example.

Step 2 - NotificationRequest (rqnt) off-hook from ca to rgw1

In this case, ca sends two rqnts, one for aaln/1:

```
rqnt 1 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567890
```

and a second for aaln/2:

```
rqnt 2 aaln/2@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567891
```

The expected response from rgw1 to these requests is an acknowledgement from aaln/1 as follows:

```
200 1 ok
```

and from aaln/2:

```
200 2 ok
```

Step 3 - AuditEndpoint (auep) from ca to rgw2

```
auep 3 *@rgw2.whatever.net mgcp 1.0
```

followed by an acknowledgement from rgw2:

```
200 3 ok
z: aaln/1@rgw2.whatever.net
z: aaln/2@rgw2.whatever.net
```

Step 4 - NotificationRequest (rqnt) from ca to each endpoint of rgw2

```
rqnt 4 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567892
```

followed by:

```
rqnt 5 aaln/2@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 234567893
```

with rgw2 acknowledging for aaln/1:

```
200 4 ok
```

and for aaln/2:

```
200 5 ok
```

G.2 Connection Creation

G.2.1 Residential Gateway to Residential Gateway

The following table shows the message sequence which occurs when a user (usr1) makes a call through a residential gateway (rgw1) to a user served by another residential gateway (rgw2). This example illustrates the communication between the residential gateways and the call agent (ca) only. The local name of the endpoints in this example is aaln/1 for both gateways, and references within the description of the steps to rgw1 and rgw2 can be assumed to refer to aaln/1 of rgw1 and aaln/1 of rgw2. Note that this is only an example and is not the only legal call scenario.

Table F.3: Residential Gateway Connection Creation

#	usr1	rgw1	ca	rgw2	usr2
1	offhook ->	ntfy ->			
			<- ack		
2	<- dialtone		<- rqnt		
		ack ->			
3	digits ->	ntfy ->			
			<- ack		
4			<- rqnt		
		ack ->			
5	<- recvonly		<- crcx		
		ack ->			
6			crcx ->		sendrcv ->
				<- ack	
7	<- recvonly		<- mdcx		
		ack ->			
8	<- ringback		<- rqnt		
		ack ->			
9			rqnt ->		ringing ->
				<- ack	
10				<- ntfy	<- offhook
			ack ->		
11			rqnt ->		
				<- ack	
12			<- rqnt		
		ack ->			
13	<- sendrcv		<- mdcx		
		ack ->			

Step 1 - Notify (ntfy) offhook from rgw1 to ca

This ntfy is the result of usr1 going offhook and assumes ca had previously sent an rqnt with RequestId "445678944" to rgw1 requesting notification in the event of an offhook:

```
ntfy 12 aaln/1@rgw1.whatever.net mgcp 1.0
o: l/hd
x: 445678944
```

Acknowledgement from ca:

```
200 12 ok
```

Step 2 - Request Notification (rqnt) for digits from ca to rgw1

Request rgw1 to notify if on-hook and collect digits according to the digit map, and to provide dialtone:

```
rqnt 1057 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n), d/[0-9#*T](d)
s: l/dl
x: 445678945
d: 5xxx
```

Acknowledgement from rgw1:

```
200 1057 ok
```

Step 3 - Notify (ntfy) digits from rgw1 to ca

```
ntfy 13 aaln/1@rgw1.whatever.net mgcp 1.0
o: d/5, d/0, d/0, d/1
x: 445678945
```

Acknowledgement from ca:

```
200 13 ok
```

Step 4 - Request Notification (rqnt) from ca to rgw1

Request rgw1 to notify in the event of an on-hook transition:

```
rqnt 1058 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678946
```

Acknowledgement from rgw1:

```
200 1058 ok
```

Step 5 - Create Connection (crcx) from ca to rgw1

Request a new connection on rgw1 with the specified local connection options, including 20 msec as the packetization period, G.711 mu-law as the codec, and receive only as the mode:

```
crcx 1059 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
l: p:20, a:PCMU
m: recvonly
```

Acknowledgement from rgw1 that a new connection, "456789fedcba5", has been created, followed by a blank line and then the SDP parameters:

```
200 1059 ok
i: 456789fedcba5

v=0
o=- 23456789 98765432 IN IP4 192.168.5.7
s=-
c=IN IP4 192.168.5.7
t=0 0
m=audio 6058 RTP/AVP 0
```

Step 6 - Create Connection (crcx) from ca to rgw2

Request a new connection on rgw2. The request includes the session description returned by rgw1 such that a two way connection can be initiated:

```
crcx 2052 aaln/1@rgw2.whatever.net mgcp 1.0
c: 9876543210abcdef
l: p:20, a:PCMU
m: sendrecv

v=0
o=- 23456789 98765432 IN IP4 192.168.5.7
s=-
c=IN IP4 192.168.5.7
t=0 0
m=audio 6058 RTP/AVP 0
```

Acknowledgement from rgw2 that a new connection, "67890af54c9", has been created; followed by a blank line and then the SDP parameters:

```
200 2052 ok
i: 67890af54c9

v=0
o=- 23456889 98865432 IN IP4 192.168.5.8
s=-
c=IN IP4 192.168.5.8
t=0 0
m=audio 6166 RTP/AVP 0
```

Step 7 - Modify Connection (mdcx) from ca to rgw1

Request rgw1 to modify the existing connection, "456789fedcba5", to use the session description returned by rgw2 establishing a half duplex connection which, though not used in this example, could be used to provide usr1 with in band ringback tone, announcements, etc:

```
mdcx 1060 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
l: p:20, a:PCMU
M: recvonly

v=0
o=- 23456889 98865432 IN IP4 192.168.5.8
s=-
c=IN IP4 192.168.5.8
t=0 0
m=audio 6166 RTP/AVP 0
```

Acknowledgement from rgw1:

```
200 1060 ok
```

Step 8 - Request Notification (rqnt) from ca for rgw1 to provide ringback

Request rgw1 to notify in the event of an on-hook transition, and also to provide ringback tone:

```
rqnt 1061 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
s: g/rt
x: 445678947
```

Acknowledgement from rgw1:

```
200 1061 ok
```

Step 9 - Request Notification (rqnt) from ca to rgw2 to provide ringing

Request rgw2 to continue to look for offhook and provide ringing:

```
rqnt 2053 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
s: l/rg
x: 445678948
```

Acknowledgement from rgw2:

```
200 2053 ok
```

Step 10 - Notify (ntfy) offhook from rgw2 to ca

```
ntfy 27 aaln/1@rgw2.whatever.net mgcp 1.0
o: l/hd
x: 445678948
```

Acknowledgement from ca:

```
200 27 ok
```

Step 11 - Request Notification (rqnt) of on-hook from ca to rgw2

```
rqnt 2054 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678949
```

Acknowledgement from rgw2:

```
200 2054 ok
```

Step 12 - Request Notification (rqnt) of on-hook from ca to rgw1

```
rqnt 1062 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hu(n)
x: 445678950
```

Acknowledgement from rgw1:

```
200 1062 ok
```

Step 13 - Modify Connection (mdcx) from ca to rgw1

Request rgw1 to modify the existing connection, "456789fedcba5", to sendrecv such that a full duplex connection is initiated:

```
mdcx 1063 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
m: sendrecv
```

Acknowledgement from rgw1:

```
200 1063 ok
```

G.3 Connection Deletion

G.3.1 Residential Gateway to Residential Gateway

The following table shows the message sequence which occurs when a user (usr2) initiates the deletion of an existing connection on a residential gateway (rgw2) with a user served by another residential gateway (rgw1). This example illustrates the communication between the residential gateways and the call agent (ca) only. The local name of the endpoints in this example is aaln/1 for both gateways, and references within the description of the steps to rgw1 and rgw2 can be assumed to refer to aaln/1 of rgw1 and aaln/1 of rgw2.

Table F.4: Residential Gateway Connection Deletion

#	usr1	rgw1	ca	rgw2	usr2
1				<- ntfy	<- on-hook
			ack ->		
2			dlcx ->		
				<- ack	
3			<- dlcx		
		ack ->			
4			rqnt ->		
				<- ack	
5	on-hook ->	ntfy ->			
			<- ack		
6			<- rqnt		
		ack ->			

Step 1 - Notify (ntfy) offhook from rgw1 to ca

This ntfy is the result of usr2 going on-hook and assumes that ca had previously sent an rqnt to rgw2 requesting notification in the event of an on-hook (see end of Connection Creation sequence):

```
ntfy 28 aaln/1@rgw2.whatever.net mgcp 1.0
o: l/hu
x: 445678949
```

Acknowledgement from ca:

```
200 28 ok
```

Step 2 - Delete Connection (dlcx) from ca to rgw2

Requests rgw2 to delete the connection "67890af54c9":

```
dlcx 2055 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 67890af54c9
```

Acknowledgement from rgw2. Note the response code of "250" meaning "the connection was deleted":

```
250 2055 ok
```

Step 3 - Delete Connection (dlcx) from ca to rgw1

Requests rgw1 to delete the connection "456789fedcba5":

```
dlcx 1064 aaln/1@rgw1.whatever.net mgcp 1.0
c: 9876543210abcdef
i: 456789fedcba5
```

Acknowledgement from rgw1:

```
250 1064 ok
```

Step 4 - NotificationRequest (rqnt) from ca to rgw2

Requests rgw2 to notify ca in the event of an offhook transition:

```
rqnt 2056 aaln/1@rgw2.whatever.net mgcp 1.0
r: l/hd(n)
x: 445678951
```

Acknowledgement from rgw2:

```
200 2056 ok
```

Step 5 - Notify (ntfy) on-hook from rgw1 to ca

Notify ca that usr1 at rgw1 went back on-hook:

```
ntfy 15 aaln/1@rgw1.whatever.net mgcp 1.0
o: l/hu
x: 445678950
```

Acknowledgement from ca:

```
200 15 ok
```

Step 6 - NotificationRequest (rqnt) offhook from ca to rgw1

Requests rgw1 to notify ca in the event of an offhook transition:

```
rqnt 1065 aaln/1@rgw1.whatever.net mgcp 1.0
r: l/hd(n)
x: 445678952
```

Acknowledgement from rgw1:

```
200 1065 ok
```

Authors' Addresses

Flemming Andreasen
Cisco Systems
499 Thornall Street, 8th Floor
Edison, NJ 08837

EMail: fandreas@cisco.com

Bill Foster
Cisco Systems
771 Alder Drive
Milpitas, CA 95035

EMail: bfoster@cisco.com

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Network Working Group
Request for Comments: 3660
Updates: 2705
Category: Informational

B. Foster
F. Andreasen
Cisco Systems
December 2003

Basic Media Gateway Control Protocol (MGCP) Packages

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

IESG Note

This document is being published for the information of the community. It describes a non-IETF protocol that is currently being deployed in a number of products. Implementers should be aware of RFC 3525 [37], which was developed in the IETF Megaco Working Group and the ITU-T SG16, and is considered by the IETF and ITU-T to be the standards-based (including reviewed security considerations) way to meet the needs that MGCP was designed to address. The IETF Megaco Working Group and the ITU-T Study Group 16 are developing extensions to RFC 3525 [37] that for functions of the type in addressed in this document.

Abstract

This document provides a basic set of Media Gateway Control Protocol (MGCP) packages. The generic, line, trunk, handset, RTP, DTMF (Dual Tone Multifrequency), announcement server and script packages are updates of packages from RFC 2705 with additional explanation and in some cases new versions of these packages. In addition to these, five new packages are defined here. These are the signal list, resource reservation, media format, supplementary services and digit map extension packages.

Table of Contents

1. Introduction 2
   1.1. List of Packages 3
   1.2. Changes to Existing RFC 2705 Packages. 3
      1.2.1. Change in Signal Types 3
      1.2.2. Operation Complete and Operation Failure 3
      1.2.3. Package Versions 4
      1.2.4. Event Definitions, Aliases and Interoperability Issues 4
      1.2.5. New Events 5
   1.3. New Packages and Excluded Packages 5
2. Packages 5
   2.1. Generic Media Package. 7
   2.2. DTMF Package 11
   2.3. Trunk Package. 16
   2.4. Line Package 24
   2.5. Handset Emulation Package. 33
   2.6. Supplementary Services Tone Package. 36
   2.7. Digit Map Extension. 37
   2.8. Signal List Package. 38
   2.9. Media Format Parameter Package 39
   2.10. RTP Package. 43
   2.11. Resource Reservation Package 48
      2.11.1. Description. 48
      2.11.2. Parameter Encoding 52
      2.11.3. Events 53
   2.12. Announcement Server Package. 55
   2.13. Script Package 56
3. IANA Considerations. 59
4. Security Considerations. 59
5. Acknowledgements 59
6. References 60
   6.1. Normative References 60
   6.2. Informative References 62
7. Authors' Addresses 63
8. Full Copyright Statement 64

1. Introduction

This document provides a basic set of Media Gateway Control Protocol (MGCP) packages. The generic, line, trunk, handset, RTP, DTMF, announcement server and script packages are updates of packages from RFC 2705 [38] with additional explanation and in some cases new versions of these packages. In addition to these, five new packages are defined here. These are the signal list, resource reservation, media format, supplementary services and digit map extension packages.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [31].

1.1. List of Packages

The basic set of packages specified in this document is for use with MGCP 1.0 as defined in [1]. Included are the following packages:

Package	Name
Generic Media Package	G
DTMF package	D
Trunk Package	T
Line Package	L
Handset Package	H
Supplementary Services Package	SST
Digit Map Extension	DM1
Signal List Package	SL
Media Format Package	FM
RTP Package	R
Resource Reservation Package	RES
Announcement Server Package	A
Script Package	Script

1.2. Changes to Existing RFC 2705 Packages

1.2.1. Change in signal types

MGCP 1.0, as defined in RFC 2705 [38] (and now updated in [1]), provided some additional clarification on the meaning of On-Off (OO) signals compared to earlier versions of MGCP. This lead to some inconsistency in some of the signal definitions in the accompanying packages in RFC 2705 [38]. This has been corrected in the packages that are included here by changing some of the signals from type On-Off to type Time-Out (TO).

1.2.2. Operation Complete and Operation Failure

Another change made to improve consistency and interoperability was to add the "operation complete" and "operation failure" events in packages where there are TO signals defined, but where the "operation complete" and "operation failure" events were not previously included as part of the package. By definition, all packages that contain

Time-Out type signals now contain the "operation failure" ("of") and "operation complete" ("oc") events as defined in [1], irrespective of whether they are provided as part of the package description or not.

If a package without Time-Out signals contains definitions for the "oc" and "of" events, the event definitions provided in the package may over-ride those indicated here. Such practice is however discouraged and is purely allowed to avoid potential backwards compatibility problems.

It is considered good practice to explicitly mention that the "oc" and "of" events are supported in accordance with their default definitions. If no definition is included in the package, the default syntax and semantics are assumed.

Please refer to [1] for additional details on these events.

1.2.3. Package Versions

The generic, line, trunk, handset, RTP, DTMF, announcement server and script packages included in this document are new versions of packages that were previously contained in RFC 2705 [38]. The updated base MGCP 1.0 specification [1] provides an optional capability of auditing package versions. Any gateway that implements versioned packages SHOULD also implement this option.

1.2.4. Event Definitions, Aliases and Interoperability Issues

Some event definitions or clarifications of previous event definitions have also been added in order to improve interoperability.

In some cases, events have aliases either in the same or in other packages and a recommendation has been made for the use of alternates by Call Agents for future implementations. For maximum interoperability, gateways MUST still implement these events (in fact they MUST always implement all of the events, signals, etc. in a package).

Some events that were previously defined require specific provisioning in both the gateway and the Call Agent in order to allow for interoperability. In those cases, a warning to that affect has been included.

1.2.5. New Events

In some cases, new events have been added to existing packages. Any changes to existing packages of course have resulted in the package version number being updated from unversioned (version 0) to version 1.

1.3. New Packages and Excluded Packages

Two packages from RFC 2705 [38] have not been included. These are the "MF" and the "NAS" packages. These packages are still valid as are all unversioned (version 0) packages defined in RFC 2705 [38]. The reason these packages were not included are:

* The original MF package had no defined way to outpulse MF digits so that MF CAS is now provided by other packages (i.e., the "MS", "MO" and "MD" packages) in a separate document.

* The "N" package, as defined in RFC 2705 [38], was incomplete. A new MGCP "NAS" package has been developed and provided in a separate document.

New packages have also been included beyond what was included in RFC 2705 [38]. These are the signal list, resource reservation, media format, supplementary services and digit map extension packages. The Resource Reservation ("RES") and Media Format ("FM") packages in particular are different from other packages in this document in that they contain new LocalConnectionOptions. This is allowed by the new extension rules in [1]. Future packages of this type MUST use a packages prefix in front of local connection options ("<package-name>/<Local Connection Option>") so as to avoid name-space problems. However because of the timing of the arrival of these packages relative to updating MGCP 1.0, this was not done for the "RES" and "FM" packages. The resulting new local connection options have been registered with IANA. For future cases where a package prefix is included, only the package name needs to be registered.

2. Packages

For those packages that involve MGCP events, the terms "signal" and "event" are used to differentiate a request from a Call Agent to a Media Gateway to apply an event ("signal"), from the request for the detection of an "event" that occurs on the Media Gateway and is "Notified" to the Call Agent.

For packages that involve events and signals, the tables contain five columns:

Symbol: the (package) unique symbol used to identify the event.

Definition: a short description of the event.

R: an x appears in this column if the event can be requested by the Call Agent. Alternatively, one or more of the following symbols may appear. An "S" is included if the event-state may be audited. A "C" indicates that the event can be detected on a connection, and a "P" indicates the event is persistent.

S: if nothing appears in this column for an event, then the event cannot be signaled by the Call Agent. Otherwise, the following symbols identify the type of event:

* OO On/Off signal

* TO Time-Out signal.

* BR Brief signal.

In addition, a "C" will be included if the signal can be generated on a connection.

Duration: specifies the default duration of TO signals. If a duration is left unspecified, then the default timeout will be assumed to be infinite, unless explicitly noted in the description of the signal. A duration may also be declared as being variable in a case where signals involve complex sequencing (e.g., scripts or digit out-pulsing) where the amount of time may vary with either processing time or the signaling environment.

Default time-out values may be over-ridden by the Call Agent for any Time-Out event defined in this document (with the exception of those that have a default value of "variable") by a "to" signal parameter which specifies the timeout value in milliseconds (see [1]). The following example indicates a timeout value of 20 seconds:

```
S: sst/cw(to=20000)
```

As indicated in [1]: by default, a supplied time-out value MAY be rounded to the nearest non-zero value divisible by 1000, i.e., whole second. However, individual signal definitions within a package may define other rounding rules.

Note that Time-Out signals that involve other parameters still allow the use of the "to" signal parameter e.g.:

S: T/sit(1,to=3000)

The order of the "to" parameter relative to the other parameters is not important.

Note: as per [1], On-Off (OO) signals are parameterized with "+" (meaning turn on) or "-" (meaning turn off). If the parameter is missing, the default is to turn on the signal. Unlike Time-Out signals, On-Off signals do not stop when an event occurs.

Other than the "to" parameter for Time-out (TO) signals and the "+" and "-" for On-Off (OO) signals, signals and events in the packages in this document do not have parameters unless explicitly indicated in the description of the event for that package.

In some of the signal definitions below, specific tone definitions are provided even though actual frequencies may vary from country to country.

2.1. Generic Media Package

Package Name: G
Version: 1

The generic media package groups the events and signals that can be observed on several types of endpoints, such as trunk gateway endpoints, access gateway endpoints or residential gateway endpoints.

Symbol	Definition	R	S	Duration
cf	Confirm Tone		BR	
cg	Congestion Tone		TO	infinite
ft	Fax Tone	x		
it	Intercept Tone		TO	infinite
ld	Long Duration Connection	C		
mt	Modem Tone	x		
oc	Operation Complete	x		
of	Operation Failure	x		
pat(###)	Pattern Detected	x	OO	
pt	Preemption Tone		TO	infinite
rbk(...)	Ringback		TO,C	180 seconds
rt	Ringback Tone		TO,C	180 seconds

New events added to this package from the previously unversioned package: "oc"

Changes: "it" and "pt" signals changed from OO to TO.

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The events and signals are defined as follows:

Confirmation Tone (cf):
This is also referred to as "positive indication tone" in ITU-T E.182. In North America, 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 [7] 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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Congestion Tone (cg):
Refer to ITU-T E.180 [8] and E.182 [10]. This maps to re-order tone in North America (refer to GR-506-CORE [7] Section 17.2.7).

Fax Tone (ft):
The fax tone event is generated whenever a fax call is detected by the 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 [21] and V.21 [22].

Intercept Tone(it):
This is a country specific tone as defined in ITU-T E.180 Supplement 2 [9].

Long Duration Connection (ld):
The "long duration connection" is detected when a connection has been established for more than a provisioned amount of time. The default value is 1 hour.

This event is detected on a connection. When no connection is specified as part of the request, the event applies to all connections for the endpoint, regardless of when the connections are created. The "all connections" wildcard (see [1]) may also be used for this case, and is in fact preferred for consistency. In

either case, the name of the connection on which the event was detected will be included when the event is observed, e.g.:

G/ld@0A3F58

Modem Tone (mt):
Indicates V.25 Answer tone (ANS) with or without phase reversals or V.8 Modified Answer Tone (ANSam) tone with or without phase reversals. Note that this implies the presence of a data call. Also note that despite the name of the event, devices other than modems may generate such tones, e.g., a fax machine.

Operation Complete (oc):
The standard definition of operation complete [1].

Operation Failure (of):
The standard definition of operation failure [1].

Pattern Detected (pat(###)):
This event requires special provisioning that needs to be agreed on between the Call Agent and media gateway in order to ensure interoperability. It is retained in order to maintain backwards compatibility with version 0 of the "G" package. This event MUST be parameterized with a decimal numeric value from 0 to 999 specifying the pattern to detect. When reported, the pattern is also included as a parameter.

Preemption Tone (pt):
This is a country specific tone and is defined in ITU-T E.180 Supplement 2 [9].

Ringback (rbk(connectionID)):
This is an alias for "rt@connectionID" and is included here for backwards compatibility only. It is recommended that Call Agents use "rt@connectionID" instead of "rbk(connectionID)" for ring-back over a connection for new implementations. Although the ringback signal is applied on a connection, the "rbk" signal does not support the "@connection" syntax. When the signal is requested, it MUST be parameterized with a connection-ID or a connection-ID wildcard as specified in [1].

Ringback Tone (rt):
Refer to ITU-T E.180 [8] and ITU-T E.182 [10]. Also referred to as ringing tone - a tone advising the caller that a connection has been made and that a calling signal is being applied to the called party or service point. In North America, this 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.

The cadence for Audible Ring Tone is 2 seconds on, followed by 4 seconds off. See GR-506-CORE [7] - LSSGR: SIGNALING, Section 17.2.5.

This signal can be applied directly to an endpoint or alternatively on a connection using the syntax "rt@connectionID". When the ringback signal is applied to an endpoint, it is considered an error to try and play ringback tone if the endpoint is considered on-hook, and an error MUST 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.

Note that as specified in [1], signals requested on a connection MUST be played regardless of the connection mode. For example, in a call-waiting situation, ringback tone may be played on a connection in "inactive" mode.

2.2. DTMF package

Package name: D
Version: 1

Symbol	Definition	R	S Duration
0	DTMF 0	x	BR
1	DTMF 1	x	BR
2	DTMF 2	x	BR
3	DTMF 3	x	BR
4	DTMF 4	x	BR
5	DTMF 5	x	BR
6	DTMF 6	x	BR
7	DTMF 7	x	BR
8	DTMF 8	x	BR
9	DTMF 9	x	BR
#	DTMF #	x	BR
*	DTMF *	x	BR
A	DTMF A	x	BR
B	DTMF B	x	BR
C	DTMF C	x	BR
D	DTMF D	x	BR
DD(..)	DTMF Tone Duration	x	TO 3 seconds
DO(..)	DTMF OO Signal		OO
L	Long Duration Indicator	x	
oc	Operation Complete	x	
of	Operation Failure	x	
T	Interdigit Timer	x	TO 16 seconds
X	DTMF Tones Wildcard, match any digit 0-9	x	

Changes from the previous version of the package: events "dd", "do", "oc" were added.

Note that DTMF tones including the DTMF tones wildcard can use the eventRange notation defined in [1] when requesting events, e.g., "D/[0-9](N)".

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The events are defined as follows:

DTMF tones (0-9,#,*,A,B,C,D):
Detection and generation of DTMF tones is described in GR-506-CORE [7] - LSSGR: SIGNALING, Section 15. Note that it is considered an error to try and play DTMF tones on a phone that is on-hook and an error MUST consequently be returned when such attempts are made (error code 402 - phone on-hook). The event codes can be specified in a digit map. When requested as a signal, as per GR-506-CORE [7], section 15, a minimum tone duration of 50 ms will be followed by a minimum interdigit silence period of 45 ms, i.e., if requested in a signal list such as "S: sl/s(d/5,d/6,d/7)", then interdigit timing requirements will be satisfied.

Note that some types of endpoints, such as announcement endpoints, MAY allow detection and/or generation of a DTMF tone over a connection. However, this requires consistent provisioning between the Call Agent and announcement server (it is not required in order to be compliant with the DTMF package).

DTMF Tone Duration (dd(dg=<tone>,to=<time>,su=<TrueOrFalse>)):
This event can be used to indicate if/when the specified <tone> has a duration greater than the <time> value indicated (and is reported once the duration is exceeded). The parameters can be supplied in any order. The value of <tone> can be any of the DTMF tone symbols (without including the package name) specified in the DTMF package (including X in the case of events, but not signals). If this parameter is absent, any DTMF tone that occurs will be reported. The parameter <time> is in milli-seconds and may be rounded to the nearest 10 ms by the gateway. The minimum value of <time> that can be requested when requesting an event is 40 ms. When requesting a signal, the minimum value of <time> that can be requested is 50 ms. The maximum value of <time> that can be requested for either an event or a signal is 60000 ms. If the "to=<time>" parameter is absent when requested as an event, the event will report the full duration (up to 60000 ms) of the tone when the tone is completed. When reported as an ObservedEvent, both parameters are always supplied. In this case, <tone> is the actual tone detected and <time> is either:

* The <time> specified in the request (possibly rounded), or

* If the request did not contain a "to=<time>" parameter, the full duration of the tone.

The parameter "su" MAY be included when this is requested as an event (but is not reported). This parameter is used to indicate whether or not the DTMF digits requested should be suppressed

in-band when it is requested. Possible values are "true", indicating that in-band DTMF should be suppressed and "false" indicating that DTMF should continue to be passed in-band. The default value of the parameter, if missing, is "false". The "su" parameter MUST NOT be included when requesting "D/dd" as a signal.

When used as a signal, "dd" provides the ability to generate a DTMF tone as a TO signal. When applied as a signal, an additional 50 ms of silence will be tacked onto the end before the operation complete occurs, i.e., "S: dd(dg=5,to=2500)" will play the DTMF tone for the number "5" for 2.5 seconds, followed by 50 ms of silence period. The operation complete (if requested) will be notified after the silence interval occurs. Any value from 50 ms to 60000 ms can be requested. Gateways generating or detecting the tone may round off the requested time to the nearest 10 ms.

The "dd" event can be used in place of the "long duration" event in order to detect a digit pressed for longer than 2 seconds. For example, in order to detect if a user presses the long "#" for longer than 2 seconds, a request could be made with the RequestedEvents line "R: d/dd(N)(dg=#,to=2000)". The resulting ObservedEvents line would be "O: d/dd(dg=#,to=2000)".

Suppose instead, that the RequestedEvents line contains

```
R: d/[0-9*#],d/dd
```

Suppose the user then pushes the "#" for 2.5 seconds. In this case, two events will be notified:

```
O: d/#
```

when the "#" key is first pressed, and

```
O: d/dd(dg=#,to=2500)
```

when the "#" key is finally released.

DTMF OO Signal (do(dg=<tone>,<on-or-off>)):
This signal is used to generate a DTMF tone as an on-off signal. The <tone> parameter is any of the symbols for a specific tone in the DTMF package (i.e., "0" to "9", "A", "B", "C", "D", "*", or "#"). The <on-or-off> indicator is "+" for on and "-" for off as per [1]. The <tone> parameter MUST be supplied, otherwise a return code of 538 - "Event/signal parameter error" will be provided in the response. If the <on-or-off> parameter is missing, the default is to turn the signal on as usual (i.e., "+" is the default). The order of the parameters is not significant

since "+" and "-" are reserved characters and are easily distinguished from the <tone> parameter.

Long Duration Indicator (l):
The "long duration indicator" is observed when a DTMF signal is produced for a duration larger 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.

Operation Complete (oc):
This is the standard definition of operation complete [1].

Operation Failure (of):
This is the standard definition of operation failure [1].

Timer (t):
Timer T can be used as an event or as a time-out (TO) signal. As a signal, its only behavior is the normal characteristics of a "TO" signal as defined in [1] (i.e., if no event occurs before the time-out, an operation complete event will be generated).

As an event, Timer T is a digit input timer that can be used in two ways:

* When timer T is used with the accumulate according to digit map action, the timer is not started until the first DTMF tone is entered, and the timer is restarted after each new DTMF tone is entered until either a digit map match or mismatch occurs. In this case, timer T functions as an inter-digit timer as illustrated by:

```
R: D/[0-9T](D)
```

* When timer T is used without the "accumulate according to digit map" action, the timer is started immediately and simply cancelled (but not restarted) as soon as a DTMF tone is entered. In this case, timer T can be used as an inter-digit timer when overlap sending is used, as in:

```
R: D/[0-9](N), D/T(N)
```

When used with the "accumulate according to digit map" action, timer T takes on one of two values, T-partial or T-critical. When at least one more symbol is required for the "current dial string" to match any one of the patterns in the digit map, timer T takes on the value T-partial, corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes

on the value T-critical corresponding to critical dial timing. When timer T is used without the "accumulate according to digit map" action, timer T takes on the value T-critical. The default value for T-partial is 16 seconds and the default value for T-critical is 4 seconds. The provisioning process may alter both of these. If timer T is not used, then inter-digit timing will not be performed.

The following examples illustrate this. Consider the digit map:

```
(xxxxxxx|x11T)
```

and assume that DTMF and the timer T is accumulated according to digit map. At the first DTMF input, say "4", timer T is started with a value of T-partial since at least one more symbol is required. If "1" is then input, it leads to a restart of timer T with a value of T-partial again. If "1" is now input again, we have a current dial string of "411" and a timer is now all that is required to produce a match. Hence timer T is now restarted with value T-critical.

Finally, consider the following subtle examples (all assuming DTMF and timer T being accumulated according to digit map):

The digit map

```
(1[2-3T].)
```

will match immediately on the input "1" since zero or more matches of the range are specified.

The digit map

```
(1[2-3].T)
```

and an input of "1" will lead to timer T being set to T-critical.

A digit map of

```
(1[2-3]T.)
```

and an input of "1" will lead to timer T being set to T-partial. Furthermore, upon subsequent input of "2" or "3" a perfect match will be triggered immediately since timer T is completely irrelevant.

DTMF Tones Wildcard (X):
The DTMF tones wildcard matches any DTMF digit between 0 and 9. The actual event code generated will however be the event code for the digit detected. The DTMF tones wildcard is often used to detect DTMF input to be matched against a digit map.

2.3. Trunk Package

Package Name: T
Version: 1

Symbol	Definition	R	S Duration
as	Answer Supervision	x	BR
bl	Blocking		BR
bz	Busy		TO 30 sec.
co1	Continuity Tone (go tone, or return tone)	x	TO 3 sec.
co2	Continuity Test (go tone, or return tone in dual tone procedures)	x	TO 3 sec.
ct(...)	Continuity Transponder		OO
lb	Loopback		OO
nm	New Milliwatt Tone	x	TO 3 sec
mm	Newest Milliwatt Tone	x	TO 3 sec
oc	Operation Complete	x	
of	Operation Failure	x	
om	Old Milliwatt Tone	x	TO 3 sec
pst	Permanent Signal Tone		TO infinite
qt	Quiet Termination		TO infinite
ro	Reorder Tone	x	TO 30 sec.
sit(#)	Special Information Tone	x	TO 2 sec. (see notes)
tl	Test Line	x	TO infinite
tp(###)	Test Pattern	x	TO 3 sec
zz	No Circuit	x	TO 2 sec

New events added to this package from the previously unversioned package: "bz", "ct", "mm", "oc", "pst", "qt", "sit", and "tp".

Changes in event types: "co1", "co2", "nm", "om", "tl", "zz" signals changed from OO to TO; "as" and "bl" changed from OO to BR.

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The definition of the trunk package events are as follows:

Answer Supervision (as):
This event is used to indicate the occurrence answer supervision. In most cases, it is a result of a steady off-hook in response to a call request. This event is included for backwards compatibility with the previous version of the package. The preferred alternative is to use the "answer" event in the appropriate CAS packages [34] (Note: check the details on the use of "answer" in the particular CAS package; in most cases "answer" as an event is an indication of a steady off-hook regardless of whether or not it is an indication of answer supervision). For details on when answer supervision is appropriate refer to [5].

Blocking (bl):
This event is used to indicate an incoming off-hook for the purposes of blocking a one-way trunk in CAS trunks. This event is included for backwards compatibility with the previous version of the package. The preferred alternative is the "block" event in the appropriate CAS packages [34].

Busy Tone (bz):
Refer to ITU-T E.180 [8]. In North America, 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, then repeating. See GR-506-CORE [7]- LSSGR: SIGNALING, Section 17.2.6.

Continuity Tone (co1):
A tone at 2010 Hz (see section 3.1.1.3 of [2]). When generated as a signal, the frequency of the tone must be within + or - 8 Hz, while the frequency of the tone corresponding to the event must be within + or - 30 Hz.

Continuity Test (co2):
A tone at 1780 Hz (see section 3.1.1.3 of [2]). When generated as a signal, the frequency of the tone must be within + or - 20 Hz, while the frequency of the tone corresponding to the event must be within + or - 30 Hz.

In continuity testing the tone corresponding to the signal at the originating gateway is referred to as the "go" tone, and the tone

corresponding to the event at that same gateway is referred to as the "return" or "check" tone.

Note that generation and notification of continuity tones are done as per continuity test requirements as defined in ITU-T Q.724 [3], as well as by Bellcore GR-317-CORE [2] specifications, i.e., the semantics of notification of the return tone is more than that the tone was received, but is an indication that the test has passed. Details are provided in the following paragraphs.

The continuity tones represented by co1 and co2 are used when the Call Agent wants to initiate a continuity test. There are two types of tests, single tone and dual tone; In the case of the dual tone, either tone can be sent and the opposite received depending on the trunk interconnections (4-wire or 2-wire) as indicated below:

```
Originating                                 Terminating
============                                ===========

   4w   --------------- 1780 Hz ------------>  2w
        <-------------- 2010 Hz -------------  (transponder)

   2w   --------------- 2010 Hz ------------>  2w/4w
        <-------------- 1780 Hz -------------  (transponder)

   4w   --------------- 2010 Hz ------------>  4w
        <-------------- 2010 Hz -------------  (loopback)
```

The Call agent is expected to know, through provisioning information, which test should be applied to a given endpoint. As an example, for a 4-wire to 2-wire connection, the Call Agent might send a request like the following to an originating gateway:

```
RQNT 1234 ds/ds1-1/17@tgw2.example.net
X: AB123FE0
S: t/co1
R: t/co2,t/oc,t/of
```

On a terminating side of a trunk, the call agent may request a continuity test connection (connection mode "conttest") to the terminating gateway as follows:

```
CRCX 3001 ds/ds1-2/4@tgw34.example.net
C: 3748ABC364
M: conttest
```

Alternatively, rather than using a connection mode, the "T/ct" signal can be used (see description of this signal further below):

```
RQNT 3001 ds/ds1-2/4@tgw34.example.net
X: 1233472
S: t/ct(in=co1,out=co2,+)
```

The originating gateway would send the requested "go" tone, and would look for the appropriate "return tone". Once the return tone is received, the originating gateway removes the go tone and checks to see that the return tone has been removed within the specified performance limits (i.e., GR-246-CORE, T1.113.4, Annex B [23]). When it detects that the test is successful, the gateway will send a notification of the return tone event (Note that notification of the return tone event therefore must not be sent prior to detection of the removal of the return tone).

The "T/co1" and "T/co2" signals are TO signals so that an operation complete event will occur when the signal times out. If a timeout value other than the default is desired, the "to" parameter may be used (e.g., "S: T/co1(to=2000)").

If the gateway detects the failure of the continuity test prior to the timeout, an operation failure event will be generated. Otherwise, the failure of the continuity test is determined by the failure to receive the return tone event before the timeout occurs (operation complete event). As with TO signals in general, operation complete and operation fail events are parameterized with the name of the signal.

In the example above where the go tone is "co1" and the return tone is "co2":

* A notification of the "co2" event indicates success (i.e., "O: t/co2").

* A notification of the operation failure event indicates failure prior to timeout (i.e., "O: t/of(t/co1)").

* A notification of the operation complete event indicates that the return tone was not received properly prior to the occurrence of the timeout (i.e., "O: t/oc(t/co2)").

On a terminating end of a trunk, either a "loopback" connection (single tone test) or "conttest" connection (dual tone test) is made (or alternatively the "T/lb" or "T/ct" signals are requested). It is up to the termination end to make sure that the return tone is removed as soon as the go tone disappears. The

Call Agent requests the removal of "contest" or "loopback" connections (or "T/lb" or "T/ct" signals) at a termination end when the results of the continuity test are obtained.

When "conttest" is used, the endpoint is provisioned as to which transponder test is being performed (2010 Hz received and 1780 Hz sent or vice versa). In the case of the corresponding "T/ct" signal, the Call Agent can specify which tone is received and sent as parameters.

Note that continuity tones in the trunk package are only ever sent to the telephony endpoint. For network-based continuity, there are continuity tones available in the RTP ("R") package. Although a transponder (dual tone) test can be done, a single tone test is generally sufficient in the case of continuity testing across an IP network.

Continuity Transponder(ct(in=<tone-in>,out=<tone-out>, <+ or ->)):
This signal is used to provide transponder functionality independent of the connection mode, i.e., this is an alternative way to provide the same functionality as the "conttest" connection mode. The parameters can be provided in any order. The <tone-in> and <tone-out> parameters can have values "co1" or "co2", corresponding to the 2010 Hz and 1780 Hz tones associated with those symbols. If one of the tones is "co1", then the other must be "co2" and vice versa (i.e., <tone-in> and <tone-out> must have different values; if loopback is required, then the "lb" signal in this package or "loopback" connection mode should be used).

On detecting <tone-in>, <tone-out> will be generated in return. The tone corresponding to <tone-out> will continue to be generated until either:

* The signal is explicitly turned off (e.g., "S: t/ct(-)") or

* Removal of the <tone-in> tone is detected.

Note that while the signal is active (regardless of whether a tone is active or not), media from the endpoint will not be forwarded to or from the packet network (i.e., the continuity transponder signal must be explicitly turned off by the Call Agent in order to resume passing media between the packet network and the endpoint).

Loopback (lb):
This signal is used to provide loopback functionality independent of the connection mode, i.e., this is an alternative way to provide the same functionality as "loopback" connection mode.

Note that while the loop-back signal is active (regardless of whether a tone is active or not), media from the endpoint will not be forwarded to or from the packet network (i.e., the loopback signal must be explicitly turned off by the Call Agent in order to resume passing media between the packet network and the endpoint).

New Milliwatt Tone (nm):
1004 Hz tone - refer to [4] and section 8.2.5 of [5].

Newest Milliwatt Tone (mm):
1013.8 Hz - refer to [4].

Operation Complete (oc):
This is the standard definition of operation complete [1].

Operation Failure (of):
This is the standard definition of operation failure [1].

Old Milliwatt Tone (om):
1000 Hz tone - refer to [4] and section 8.2.5 of [5].

Permanent Signal Tone (pst):
In North America, this tone is applied to a busy line verify/operator interrupt under specific circumstances as described in [17].

Quiet Termination (qt):
Quiet Termination is used in a 102 trunk test. Reference section 6.20.5 [5] as well as [4].

Reorder Tone(ro):
This maps to congestion tone in the ITU-T E.182 specification. In North America, 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 (until time-out). See GR-506-CORE [7], Section 17.2.7.

Special Information Tone(sit(#)):
As described in ITU-T E.180 [8], the special information tone consists of a tone period in which 3 tones are produced followed by a silent period of 1 second (total TO period of approximately 2 seconds). When used as a signal, it MUST be parameterized with a parameter value from 1 to 7, with the following meaning as defined in SR-2275, section 6.21.2 of [5].

sit(1)	RO'	reorder SIT, intra-LATA
sit(2)	RO"	reorder SIT, inter-LATA
sit(3)	NC'	no circuit SIT, intra-LATA
sit(4)	NC"	no circuit SIT, inter-LATA
sit(5)	IC	intercept SIT
sit(6)	VC	vacant code SIT
sit(7)	IO	ineffective other SIT

When requested as an event, the event MUST be parameterized with a decimal number from 1 to 7 to indicate which tone the gateway is required to detect. The resulting notification also includes the parameter. Other countries may have one or more special information tones with country specific definitions (refer to ITU-T E.180 supp. 2 [9]). In this case, special information tone 1 as defined in [9] is sit(1), special information tone 2 is sit(2) etc.

As an example, the Call Agent might make a request such as:

```
RQNT 1234 ds/ds1-1/17@tgw2.example.net
X: AB123FE0
R: t/sit(N)(2)
```

If the tone is detected, the resulting notification might appear as follows:

```
NTFY 3002 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: AB123FE0
O: t/sit(2)
```

Test Line (tl):
105 Test Line test progress tone (2225 Hz + or - 25 Hz at -10 dBm0). Refer to section 8.2.5 of [5].

Test Pattern (tp(###)):
The tp(###) signal inserts the pattern ### continuously into the channel until the timeout period expires. The parameter is provided as a decimal number from 0 to 255. If the parameter is omitted, the default value is decimal 95.

In RequestedEvents, the parameter MAY be supplied to indicate what pattern the Call Agent wishes the gateway to detect. If the parameter is omitted, the value 95 is assumed. The pattern MUST be returned in the ObservedEvent (even if the parameter was not requested).

A typical use for the test pattern signal is for the test line 108 (digital loopback) test (refer to section 8.2.5 of [5]). At the termination side of a trunk, the Call Agent would request a connection in "loopback" mode, which would do a digital loopback. On the origination side of the trunk, the Call Agent would request that the test pattern be injected into the digital channel, and would check to see that the pattern was returned within the timeout period. As an example, the Call Agent would make the following request on the origination side:

```
RQNT 1234 ds/ds1-1/17@tgw2.example.net
X: AB123FE0
S: t/tp
R: t/tp, t/oc, t/of
```

In this case the Call Agent will either receive:

* An ObservedEvent indicating that the test has passed (i.e., "O:t/op(95)") or

* An ObservedEvent indicating that the timeout occurred before the pattern was received (i.e., "O:t/oc(t/tp)"), indicating that the test failed. Of course an operation failure would indicate failure as well.

No Circuit (zz):
This is an alias for Special Information Tone 2, i.e., "sit(2)".

2.4. Line Package

Package Name: L
Version: 1

Symbol	Definition	R	S Duration
adsi(string)	ADSI Display		BR
aw	Answer Tone	x	OO
bz	Busy Tone		TO 30 sec.
ci(ti,nu,na)	Caller-id		BR
dl	Dial Tone		TO 16 sec.
e	Error Tone	x	TO 2 sec.
hd	Off-hook Transition	S	
hf	Flash-hook	x	
ht	On Hold Tone		OO
hu	On-hook Transition	S	
lsa	Line Side Answer Sup.		OO
mwi	Message Waiting ind.		TO 16 sec.
nbz	Network busy	x	TO infinite
oc	Operation Complete	x	
of	Operation Failure	x	
osi	Network Disconnect		TO 900 ms
ot	Off-hook Warning Tone		TO infinite
p	Prompt Tone	x	BR
rg	Ringing		TO 180 sec.
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive Ringing		TO 180 sec.
ro	Reorder Tone		TO 30 sec.
rs	Ringsplash		BR
s(###)	Distinctive Tone Pattern	x	BR
sit(#)	Special Information Tone		TO 2 sec. (see notes)
sl	Stutter Dial Tone		TO 16 sec.
v	Alerting Tone		OO
vmwi	Visual Message Waiting Indicator		OO
wt	Call Waiting Tone		TO 12 sec
wt1, wt2, wt3, wt4	Alternative Call Waiting Tones		TO 12 sec (see notes)
y	Recorder Warning Tone		TO infinite
z	Calling Card Service Tone		BR

New events added to this package from the previously unversioned package: "ht", "osi", and "lsa".

Changes in event types: signals "y", "z", changed from OO to TO and BR respectively. Ringing tones were extended to allow for a ring repetition signal parameter.

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The description of events and signals in the line package are as follows:

ADSI Display (adsi):
This signal is included here to maintain compatibility with the previous version of this package. The signal is not well-defined and its use is discouraged.

Answer Tone (aw):
This event is included here to maintain compatibility with the previous version of this package. The event is not well-defined and its use is discouraged.

Busy Tone (bz):
Refer to ITU-T E.180 [8]. In North America, 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 [7], 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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Caller-id (ci(time, number, name)):
See GR-1188 [24], GR-30-CORE [14], and GR-31 [25]. For backwards compatibility, each of the three fields are optional, but each of the commas will always be included. In accordance with the general MGCP grammar, it is RECOMMENDED to always include all three fields - an empty quoted string can then be used in lieu of omitting a parameter:

The time parameter is coded as "MM/DD/HH/MM", where MM is a two-digit decimal value for a Month between 01 and 12, DD is a two-digit value for a Day between 01 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. (Note: two digits MUST always be provided for each of the values of month, day, hour, minutes e.g., the month of January is indicated by the two digits "01" rather than just "1").

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, but they will be ignored. If a quoted-string is provided, the string itself is UTF-8 encoded (RFC 2279) as usual for signal parameters.

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. If a quoted-string is provided, the string itself is UTF-8 encoded (RFC 2279).

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. Other letters MAY be used to provide additional clarification as per provider or vendor specifications.

The following example illustrates the use of the caller-id signal:

```
S: l/ci(09/14/17/26, "555 1212", "John Doe")
```

An example indicating that the name and number are private:

```
S: l/ci(09/14/17/26,P,P)
```

Dial Tone (dl):
Refer to the ITU-T E.180 [8] specification. In North America, 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 [7] - 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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Error Tone (e):
This tone is maintained for backwards compatibility. The tone is not well defined and its use is discouraged.

Off-hook Transition (hd):
See GR-506-CORE [7], Section 12. It is considered an error to try and request off-hook on a phone that is off-hook and an error MUST consequently be returned when such attempts are made (error code 401 - phone off-hook).

Flash Hook (hf):
See GR-506-CORE [7], Section 12. It is considered an error to try and request flash hook on a phone that is on-hook and an error MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Tone On Hold (ht):
A tone used to reassure a calling subscriber who has been placed on "hold". Refer to ITU-T E.182 [10].

On-hook Transition (hu):
See GR-506-CORE [7], Section 12. The timing for the on-hook signal is for flash response enabled, unless provisioned otherwise. It is considered an error to try and request flash hook on a phone that is on-hook and an error MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Line Side Answer Supervision (lsa):
This provides Reverse Loop Current Feed (RLCF) on the line (refer to GR-506-CORE [7]) and is a way of indicating that the called party has answered for some line-side equipment.

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 [7], 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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Network Busy (nbz):
This is included here to maintain compatibility with the previous version of this package. The "nbz" signal is an alias for re-order tone signal("ro"). Future Call Agent implementations that require a network busy signal should use the "ro" signal. It is also recommended that future Call Agents not request to be notified of the "nbz" event (a network busy event is generally not required in a line package; hence, "ro" is only a signal, not an event).

Operation Complete (oc):
This is the standard definition of operation complete [1].

Operation Failure (of):
This is the standard definition of operation failure [1].

Network Disconnect (osi):
Network Disconnect indicates that the far-end party has disconnected. The signal that is sent on the line is provisioned in the media gateway since it may vary from country to country. In North America, this signal is an open switch interval which results in a Loop Current Feed Open Signal (LCFO) being applied to the line (refer to GR-506-CORE [7], see also See GR-505-CORE [6], Section 4.5.2.1). The default time-out value for this signal is 900 ms.

Off-hook Warning Tone (ot):
Off-hook warning tone, also known as receiver Off-Hook Tone (ROH Tone). This is the irritating noise a telephone makes when it is not hung up correctly. In North America, 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, then repeating. GR-506-CORE [7], 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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Prompt Tone (p):
The definition of the prompt tone and its use may be found in requirement GR-220 [20]. The tone in GR-220 (requirement "R3-170" or GR-220) is a 300 ms burst of a 400 Hz tone.

Ringing (rg):
See GR-506-CORE [7], 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 value for "rep" is specified in decimal and can have any value from 1 to 255. The following will apply the ringing signal for up to 6 ringing cycles:

```
S: l/rg(rep=6)
```

If the "rep" parameter is specified, the signal times-out when the number of repetitions are completed (i.e., an operation complete event can be requested and will occur at the end of the timeout/number of rings).

If the "rep" parameter is supplied, then any timeout ("to") value that is included will be ignored, i.e.:

```
S: l/rg(rep=6,to=12000)
```

will be treated the same as the previous example where the parameter "to=12000" was not included. Of course, if the "to" parameter is included without the "rep", it will be acted upon i.e.:

```
S: l/rg(to=12000)
```

will ring for 12 seconds.

It is considered an error to try and ring a phone that is off-hook and an error MUST consequently be returned when such attempts are made (error code 401 - phone off-hook).

Distinctive Ringing (r0, r1, r2, r3, r4, r5, r6 or r7):
See GR-506-CORE [7], Section 14. Default values for r1 to r5 are as defined for distinctive ringing pattern 1 to 5 in GR-506-CORE [7]. The default values for r0, r6 and r7 are normal ringing (i.e., the same cadence "rg"). The provisioning process may define the ringing cadence for each of these signals. The distinctive ringing signals may be parameterized with the signal parameter "rep" which specifies the maximum number of ringing cycles (repetitions) to apply. The value for "rep" is specified in decimal and can have any value from 1 to 255.

The following will apply the ringing signal for up to 6 ringing cycles:

```
S: l/r1(rep=6)
```

If the "rep" parameter is specified, the signal times-out when the number of repetitions are completed (i.e., an operation complete event can be requested and will occur at the end of the timeout/number of rings)

If the "rep" parameter is supplied, then any timeout ("to") value that is included will be ignored, i.e.:

```
S: l/r1(rep=6,to=12000)
```

will be treated the same as the previous example where the parameter "to=12000" was not included. Of course, if the "to" parameter is included without the "rep", it will be acted upon i.e.:

```
S: l/r1(to=12000)
```

will ring for 12 seconds.

It is considered an error to try and ring a phone that is off-hook and an error MUST consequently be returned when such attempts are made (error code 401 - phone off-hook).

Reorder Tone (ro):
This maps to congestion tone in the ITU-T E.182 [10] specification. In North America, 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.

Ringsplash (rs):
Also known as "Reminder ring", this tone 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 Forward sub-feature is active. In the US, it is defined to be a 0.5(-0,+0.1) second burst of power ringing (see [11]).

Distinctive Tone Pattern (s(###)):
This is used to signal or detect a tone pattern defined by the parameter where the parameter may have a value from 0 to 999. When specified as an event, the parameter MUST be included. The parameter will also be included when the event is reported. This event (the definition of tones associated with each parameter value) requires special provisioning in the Call Agent and gateway to insure interoperability. This signal is included here to maintain compatibility with the previous version of this package.

Special Information Tone(sit(#)):
As described in ITU-T E.180 [8], the special information tone consists of a tone period in which 3 tones are produced, followed by a silent period of 1 second (total TO period of approximately 2 seconds). It MAY be parameterized with a parameter value from 1 to 7, with the following meaning as defined in SR-2275, section 6.21.2 [5]:

sit(1)	RO'	reorder SIT, intra-LATA
sit(2)	RO"	reorder SIT, inter-LATA
sit(3)	NC'	no circuit SIT, intra-LATA
sit(4)	NC"	no circuit SIT, inter-LATA
sit(5)	IC	intercept SIT
sit(6)	VC	vacant code SIT
sit(7)	IO	ineffective other SIT

If the parameter is left out, the NC' SIT tone that corresponds to the signal "L/sit(3)" is assumed.

Other countries may have one or more special information tones with country specific definitions (refer to ITU-T E.180 supp. 2 [9]). In this case, special information tone 1 as defined in [9] is sit(1), special information tone 2 is sit(2) etc.

Stutter Dial Tone (sl):
Stutter Dial Tone (also called Recall Dial Tone in GR-506-CORE [7] and "special dial tone" in ITU-T E.182 [10]) is used to confirm some action and request additional input from the user. An example application is to cancel call-waiting, prior to entering a destination address.

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. The parameter can have any value from 0 to 10000 ms, rounded to the nearest non-zero value divisible by 100 (i.e., tenth of a second). The following will apply stutter dial tone with a delay of 1.5 seconds between the confirmation tone and the dial tone:

```
S: l/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 MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Alerting Tone (v):
A 440 Hz Tone of a 2 second duration, followed by a 1/2 second of tone every 10 seconds. This event is included for backwards compatibility with the previous version of the package.

Visual Message Waiting Indicator (vmwi):
The transmission of the VMWI messages will conform to the requirements in [13] and the CPE guidelines in [12]. Refer also to section 6.6 of GR-30-CORE [14]. VMWI messages will only be sent from the gateway 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. After the gateway restarts, the state of the signal will be "off", and hence the Call Agent MUST refresh the CPE's visual indicator if it is supposed to be "on".

Alternative Call Waiting Tones (wt, wt1, .., wt4):
Refer to ITU-T E.180 [8]. For North American tone definitions, refer to GR-506-CORE [7], Section 14.2. "wt" and "wt1" are both aliases for the default Call Waiting tone, which in North America, is a 440-Hz tone applied for 300 plus or minus 50 ms. The tone is then repeated once after 10 seconds.

These signals are timeout signals with a default timeout value of 12 seconds, which allows the tone to be played twice with a single request (refer to GR-571-CORE [16]). However, there are cases (Requirement R3-73 of GR-575-CORE [18]), in which only a single tone is required. In that case, the Call Agent may make the request with a shorter timeout period to eliminate the second tone (e.g., "S: wt(to=2000)" - which stops the signal after 2 seconds so that the second tone will not occur).

Signals wt2, wt3 and wt4 are alternates that are used for distinctive call-waiting tone patterns (refer to GR-506-CORE, Section 14.2 [7]. It is considered an error to try and apply call-waiting tone on a phone that is on-hook and an error MUST consequently be returned when such attempts are made (error code 402 - phone on-hook).

Recorder Warning Tone(y):
Refer to ITU-T E.180 [8] - also Bellcore document SR-2275 [5] section 6.20. When recording equipment is used, this tone is connected to the line to inform the distant party that the conversation is being recorded - typical value used is a 1400 Hz Tone of a 0.5 second duration every 15 seconds.

Calling Card Service Tone(z):
This tone is used to inform the customer that credit card information must be keyed in. Typically, it consists of 60 ms of 941 + 1477 Hz (the DTMF #digit) and 940 ms of 350 + 440 Hz (dial tone), decaying exponentially with a time constant of 200 ms. Refer to Bellcore document SR-2275 [5], section 6.20.

2.5. Handset Emulation Package

Package Name: H
Version: 1

Symbol	Definition	R	S Duration
adsi(string)	ADSI Display	x	BR
aw	Answer Tone	x	OO
bz	Busy Tone	x	TO 30 sec.
ci(ti,nu,na)	Caller-id	x	BR
dl	Dial Tone	x	TO 16 sec.
e	Error Tone	x	TO 2 sec.
hd	Off-hook Transition	S	BR
hu	On-hook Transition	S	BR
hf	Flash Hook	x	BR
ht	Tone On Hold	x	OO
lsa	Line Side Answer Sup.	x	OO
mwi	Message Waiting Ind.	x	TO 16 sec.
nbz	Network Busy	x	TO infinite
oc	Operation Complete	x	
ot	Off-hook Warning Tone	x	TO infinite
of	Operation Failure	x	
osi	Network Disconnect	x	TO 900 ms
p	Prompt Tone	x	BR
rg	Ringing	x	TO 180 sec.
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive Ringing	x	TO 180 sec.
ro	Reorder Tone	x	TO 30 sec.
rs	Ringsplash	x	BR
s(###)	Distinctive Tone Pattern	x	BR
sit(#)	Sit Tone	x	TO 2 sec.
sl	Stutter Dial Tone	x	TO 16 sec.
v	Alerting Tone	x	OO
vmwi	Vis. Message Waiting Ind.	x	OO
wt	Call Waiting tone	x	TO 12 sec.
wt1, wt2, wt3, wt4	Alternative Call Waiting Tones	x	TO 12 sec (see notes)
y	Recorder Warning Tone	x	TO infinite
z	Calling Card Serv. Tone	x	BR

The handset emulation package is similar to the line package except that events such as "off-hook" can be signaled as well as detected.

Changes from the original package - are the same changes as were made for the line package, plus "hu" and "hd" signal types were changed from OO to BR.

Event definitions are the same as for the line package with the following exceptions:

ASDI:
When requested as an event by the Call Agent, the event is not parameterized. However, the parameter is included when the event is reported.

Caller-id:
When requested as an event by the Call Agent, the event MUST not be parameterized. However, parameters are included when the event is reported i.e.:

```
O: l/ci(09/14/17/26,"555 1212","John Doe")
```

Line Side Answer Supervision:
When requested as an event by the Call Agent, it indicates when the reverse loop current feed (RLCF) was turned on and off. The event is not parameterized when it is requested. However, a parameter is included when it is reported i.e.:

```
O: l/lsa(+) to indicate RLCF was turned on
O: l/lsa(-) to indicate RLCF was turned off
```

Ringing (rg):
When requested as an event, the Call Agent may optionally include the rep parameter indicating a request to report after some number of rings e.g.:

```
RQNT 1234 aaln/1@rgw2.example.net
X: AB123FE0
R: h/rg(N)(rep=3)
```

The resulting notification after the number of rings is detected includes the parameter again:

```
NTFY 3002 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: AB123FE0
O: h/rg(rep=3)
```

If the parameter is not included in the request, it is also not included in the report. In that case, the event is reported as soon as ringing is detected.

Distinctive Ringing (r0, r1, r2, r3, r4, r5, r6 or r7):
As with the "rg" event, if the "rep" parameter is included when one of these is requested as an event, it is also reported. If it is not requested with the parameter, then the parameter is also not included in the report. In that case, the event is reported as soon as ringing with the requested cadence is detected.

Stutter Dial Tone (sl):
Stutter Dial Tone MUST not be parameterized when requested as an event. However, the "del" parameter is reported.

```
RQNT 1234 aaln/1@rgw2.example.net
X: AB123FE0
R: h/sl
```

The resulting notification indicates the delay between the confirmation tone and the dial tone:

```
NTFY 3002 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: AB123FE0
O: h/sl(del=1500)
```

As with the signal, the report indicates the delay rounded to the nearest 100 ms.

Visual Message Waiting:
When requested as an event by the Call Agent, it indicates when the visual message waiting indicator was turned on and off. The event is not parameterized when it is requested. However, a parameter is included when it is reported i.e.:

```
O: l/vmwi(+) to indicate message waiting turned on
O: l/vmwi(-) to indicate message waiting turned off
```

Note that:

* All TO signals in the handset package can include a "to" parameter when requested as a signal.

* However, requests to be notified about these events MUST NOT include the "to" parameter, i.e., the "to" parameter is not valid in RequestedEvents.

2.6. Supplementary Services Tone Package

Package Name: SST
Version: 0

Symbol	Definition	R	S Duration
cd	Conference Depart		BR
cj	Conference Join		BR
cm	Comfort Tone		TO infinite
cw	Caller Waiting Tone		TO 30 sec.
ht	On Hold Tone		OO
ni	Negative Indication		TO infinite
nu	Number Unobtainable		TO infinite
oc	Operation Complete	x	
of	Operation Failure	x	
pr	Pay Phone Recognition		BR
pt	Pay Tone		BR

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The events in this package are defined as follows:

Conference Depart(cd):
Tone used to indicate that a participant has left a conference call. The tone characteristics are left to the specific gateway implementation.

Conference Join (cj):
Tone used to indicate that a party has joined a conference call. The tone characteristics are left to the specific gateway implementation.

Comfort Tone (cm):
Comfort Tone is used to indicate that the call is being processed and that the caller should wait. Refer to ITU-T E.182 [10].

Caller Waiting Tone (cw):
Not to be confused with a call-waiting tone, this is a tone advising a caller that a called station, though busy, has a call waiting service active. Refer to ITU-T E.182 [10].

Tone on-hold (ht):
A tone used to reassure a calling subscriber who has been placed on "hold". Refer to ITU-T E.182 [10].

Negative Indication (ni):
A tone advising a subscriber that the request for service cannot be accepted. Refer to ITU-T E.182 [10]. For North America, this maps to the re-order tone (see GR-506-CORE [7], Section 17.2.7).

Number Unobtainable Tone (nu):
Refer to ITU-T E.180, supplement 2 [9]. This is also referred to as "vacant tone" and maps to a "re-order tone" in North America (see GR-506-CORE [7], Section 17.2.7).

Operation Complete (oc):
The standard definition of operation complete [1].

Operation Failure (of):
The standard definition of operation failure [1].

Pay Phone Recognition (pr):
A tone advising an operator that the endpoint is identified as a payphone. Refer to ITU-T E.182 [10].

Pay Tone (pt):
A tone indicating that payment is required. Refer to ITU-T E.182 [10].

2.7. Digit Map Extension

Package Name: dm1 ("dm" followed by the number "1")
Version: 0
Extension Digit Map Letters: P

This package defines an Extension Digit Map Letter that is used to override the shortest possible match behavior for a given entry in a digit map (see [1]). The letter "P" (for partial match override), at the end of a digit map entry, instructs the gateway to only consider that entry a match if the current dial string does not partially match another entry. For example, given the digit map

```
([3-7]11|123xxxxxxx|[1-7]xxxxxxP|8xxxP)
```

and a current dial string of "1234567", we would not consider this a match (as the rules in [1] would otherwise imply); however a current dial string of "411" would be considered a match as usual. A current dial string of "8234" would be considered a match since there is no other partial match.

Note that the digit map letter "P" is not an event, but simply a syntactic and semantic digit map extension. Thus, the "P" is not included in the list of requested or observed events.

Support for this package is strongly RECOMMENDED.

2.8. Signal List Package

Package Name: SL
Version: 0

Symbol	Definition	R	S	Duration
oc	Operation Complete	x		
of	Operation Failure	x		
s(list)	Signal List		TO	variable

Operation Complete (oc):
This is the standard definition of operation complete from [1].

Operation Failure (of):
This is the standard definition of operation failure from [1].

Signal List(s(<list>)):
The <list> contains a comma-separated list of signals to be played out. Each of the signals in <list> MUST be either of type BR or type TO. Semantically, the signal list is still treated as a single parameterized signal of type Time-Out though. The signals in the list are played to completion one after the other in the left to right order specified. The package for each signal in the list must be specified. For example, to play out the DTMF digits 123456:

```
S: sl/s(d/1,d/2,d/3,d/4,d/5,d/6)
```

This will result in the DTMF digits 1, 2, 3, 4, 5 and 6 being played out in order.

It is illegal to include an OO signal as one of the signals in the list or to request recursive definitions (signal lists within signal lists). If this or any other unsupported signal is included, error code 538 (event/signal parameter error) MUST be returned by the gateway.

Note that as the gateway plays the ordered list of signals, if it encounters a TO signal with an infinite timeout, it will continue to play that signal until the Signal List signal is stopped (i.e., other signals later in the list will never be played).

If the operation complete ("oc") event is requested, it will be detected once, when the last signal in the list has been played out (regardless of whether there are any TO signals in the list). The operation complete event will only report the signal list name itself, i.e., without the parameters supplied as in:

```
O:   sl/oc(sl/s)
```

Should any of the signals in the signal list result in an error, an operation failure event for the Signal List signal MUST be generated. Only the signal list name will be included, thus it is not possible to determine which of the signals in the signal list actually failed.

Note that if an event occurs while the "SL/S" signal is playing, the "SL/S" signal is stopped in the following manner:

* If the signal in the list that was playing at the time the event occurred is of type BR, then the BR signal will be played to completion and no other signals in the list will be played.

* If the signal in the list that was playing at the time the event occurred is of type TO, then the TO signal will stop immediately and no other signals in the list will be played.

2.9. Media Format Parameter Package

Package Name: FM
Version: 0

This package provides support for the media format parameter Local Connection Option (LCO). The media format parameter LCO is similar to the "fmtp" attribute in SDP [15] and is applicable to all of the same media formats that the corresponding SDP fmtp attribute could be used with (i.e., media format parameters for any media format MIME type). The media format parameter is encoded as the keyword "fmtp" or "o-fmtp", followed by a colon and a quoted string beginning with the media format name (MIME subtype only) followed by a space, followed by the media format parameters associated with that media format. For simplicity, we will use the terms "codec" and "media

format" interchangeably in the following. Multiple formats may be indicated by either repeating the "fmtp" local connection option multiple times, such as:

```
L:a:codec1;codec2, fmtp:"codec1 formatX", fmtp:"codec2 formatY"
```

or alternatively by having a single "fmtp" keyword followed by a colon, and a semi-colon separated list of quoted strings for each media format parameter, as in:

```
L:a:codec1;codec2, fmtp:"codec1 formatX";"codec2 formatY"
```

The two formats may be mixed.

If it is possible for the same codec to be requested with and without the special "fmtp" format, the following could result:

```
L:a:codec1;codec1, fmtp:"codec1 formatX"
```

However, it would not be clear if the fmtp parameter was to be applied to the first or the second occurrence of the codec. The problem with that is, that codec ordering is important (i.e., codecs are listed in preferred order), and the above syntax does not provide a way to indicate if "formatX" is preferred (i.e., associated with the first "codec1") or not (i.e., associated with the second "codec1"). In order to resolve this dilemma, when the same codec is requested with multiple formats, the codec name in the "fmtp" format string is followed by a colon and an <order>, where <order> is a number from one to N for N occurrences of the same codec in the codec list i.e.:

```
L:a:codec1;codec1, fmtp:"codec1:2 formatX"
```

indicates that "formatX" is associated with the second instance of "codec1" in the "a:codec1;codec1" list. If an invalid instance number is supplied (e.g., instance 3 where there are only two instances), then error code 524 - inconsistency in local connection options will be returned.

Pre-pending "fmtp" with the string "o-" (i.e., "o-fmtp") indicates that the format is optional. In that case, the gateway may decide not to use the fmtp parameter specified, or only use it in part.

If the "fmtp" in an LCO is not optional (i.e., does not have "o-" in front of it), and the LCO value is either not recognized or not supported, then the associated codec is considered "not supported".

When auditing capabilities, the "fmtp" local connection option MUST be returned with a semi-colon separated list of supported formats and/or multiple independent "fmtp" parameters as in:

```
A: a:telephone-event, fmtp:"telephone-event 0-15,32-35",...

A: a:PCMU;G729, fmtp:"PCMU foo";"PCMU bar", fmtp:"G729 foobar",...
```

One example uses the media format parameter LCO in conjunction with the media format "telephone-event", as defined in RFC 2833 [33]. If the media format "telephone-event" is used without the "fmtp" media format parameter, the DTMF digits (telephone events 0-15 from RFC 2833 [33]) are assumed - such practice is however discouraged. On the other hand, the media format parameter LCO MAY be used to specify the exact set of events that are being requested via RFC 2833 [33]. Example:

```
L: a:PCMU;telephone-event,fmtp:"telephone-event 16"
```

indicates that if telephone events are supported at all, then this request is specifically for event 16.

In another case, the Call Agent may indicate that some format parameters are "required", while others are optional. In the example below, telephone events 0-15 are a "must", while telephone events 16, 70 and 71 are optional.

```
L: a:PCMU;telephone-event, o-fmtp:"telephone-event 16,70,71",
fmtp:"telephone-event 0-15"
```

If the gateway cannot support telephone events 0-15, it MUST NOT include the "telephone-event" media format in the SDP in its response. On the other hand, if it can support those telephone events, it SHOULD indicate support for those events, as well as any of the events 16, 70 and 71 that it supports.

If a request is made to audit the capabilities of an endpoint, and the endpoint supports the "telephone event" media format with events "0-16", then the audit would include the following:

```
A: a:telephone-event, fmtp: "telephone-event 0-16"
```

Another example is the use of redundancy with RFC 2198 [32]. Again, the format of the fmtp string is similar to that used in the SDP except that the literal string ("red" in this case) is used rather than the payload type:

```
L: a:G729;pcmu;red,fmtp:"red pcmu/g729"
```

The corresponding media description in the SDP as part of the connection request acknowledgment might look like:

```
m=audio 12345 RTP/AVP 98 18 0
a=rtpmap:98 red/8000/1
a=fmtp:98 0/18
```

If we combine both telephone events and redundancy, an example local connection option might look as follows (carriage return added for formatting reasons here):

```
L: a:G729;pcmu;red;telephone-event,fmtp:"red pcmu/g729",
                                   fmtp: "telephone-event 16"
```

Note that we again specify the literal string for the encoding method rather than its payload type. This is a general principle that should be used with this LocalConnectionOption.

The corresponding SDP might appear as follows:

```
m=audio 12345 RTP/AVP 97 98 18 0
a=rtpmap:97 red/8000/1
a=fmtp:97 0/18
a=rtpmap:98 telephone event
a=fmtp:98 16
```

Note that the fmtp LCO may be used in any situation where the corresponding SDP attribute may be used. An example of a local connection option that involves a media type other than audio and a "foobar" fmtp parameter:

```
L: a:image/tiff, fmtp:"tiff foobar"
```

Note that normally local connection options that are associated with a package should have the package prefix included as per the package extension rules in [1]. The "fmtp" and "o-fmtp" LCO in the "FM" package are an exception. The package prefix is not included in the case of the "fmtp" and "o-fmtp" local connection options because they were created before the extension rules in [1] were defined.

These two LocalConnectionOptions have been registered with IANA.

2.10. RTP Package

Package Name: R
Version: 1

Symbol	Definition	R	S Duration
co1	Continuity Tone (single or return tone)	C	TO,C 3 sec.
co2	Continuity Test (go tone, in dual tone procedures)	C	TO,C 3 sec.
iu(..)	ICMP Unreachable Received	C	
ji(..)	Jitter Buffer Size Changed	C	
ma	Media Start	C	
oc	Operation Complete	x	
of	Operation Failure	x	
pl(..)	Packet Loss Exceeded	C	
qa	Quality Alert	C	
rto(..)	RTP/RTCP Timeout	C	
sr	Sampling Rate Changed	C	
uc	Used Codec Changed	C	

Changes in event types: "co1" and "co2" signals changed from OO to TO.

New events added to this package from the previously unversioned package: "iu", "rto", "ma".

Note that default time-out values may be over-ridden by the Call Agent for any Time-Out signal defined in this package by a "to" signal parameter. Refer to section 2 of this document, as well as [1] for details.

The events in this package all refer to media streams (connections), i.e., they cannot be detected on an endpoint. Furthermore, with the exception of the "iu" event, which is defined for any type of media, all other events in this package are defined for RTP media streams only (i.e., if they are used on connections that do not use RTP, the behavior is not defined).

Signals requested (e.g., "co1" and "co2") must indicate the connection ID (e.g., "S: r/co1@connectionID"). An event may be requested for all existing connections using the "*" wildcard for the connectionID as described in [1].

Example:

R: r/uc@* (request to detect uc on all connections) or

R: r/uc@connectionID (request to detect uc only on a specific connection)

An event detected on a connection will include the connectionID, e.g.:

O: r/uc@connectionID(15)

Continuity tones (co1 and co2):
These are the same as the events defined in the Trunk package, except in this case, they are only played over a network connection and the connectionID MUST be supplied (e.g., "s: r/co1@connectionID"). They can be used in conjunction with the Network LoopBack (netwloop) or Network Continuity Test (netwtest) modes to test the continuity of an RTP circuit. However, in the case of testing IP continuity, a one-tone test is sufficient i.e., generating and detecting "co1" at one end, with connection mode in network loopback mode at the other end. Note that the test can also be done using telephone events rather than tones, i.e., event 167 in RFC 2833 [33] corresponds to "co1". In this case, connection requests are made with local connection options such as:

L: a:PCMU;telephone-event,fmtp:"telephone-event 167"

in order to request support for telephone event 167. If both ends support the event, then the network loopback proceeds as usual, except that telephone events corresponding to the co1 tone are sent, as opposed to the co1 tone itself.

ICMP Unreachable Received (iu):
This event indicates that some number of ICMP unreachable packets [19] was received for this connection since an RQNT was received requesting this event. This notification indicates that packets that were sent by the gateway on this connection either did not arrive at their destination or were not accepted (e.g., the port was closed). When this event is requested, a single parameter with a decimal number from 1 to 255 may be included to indicate the number of ICMP un-reachable packets that must occur before the event is notified. If no parameter is supplied, with the request then a default value of 3 is assumed. This is a one-shot event in that once the event occurs, a further request is required in order to re-initiate counting.

The observed event is parameterized with two parameters:

* The first parameter is the number of ICMP unreachable packets received (i.e., the same value that was included in the request - or the value 3, if the requested event was not parameterized)

* The second parameter is the error code indicated in the ICMP unreachable packet, e.g.:

  0 = net unreachable;

  1 = host unreachable;

  2 = protocol unreachable;

  3 = port unreachable;

  4 = fragmentation needed and DF set;

  5 = source route failed.

  etc.

An example of a request might be as follows:

```
RQNT 2001 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: 0123456789B0
R: r/iu@364823(N)(5)
```

In this case, a notify will occur if 5 ICMP port unreachable packets are received as a result of RTP and/or RTCP packets being sent from this gateway on the connection with connection ID 364823.

The resulting NTFY with observed events might be as follows:

```
NTFY 3002 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: 0123456789B0
O: r/iu@364823(5,3)
```

The first parameter indicates 5 ICMP unreachable packets were received since the RQNT with this request was sent. The second parameter ("3") specifies the reason, which in this case, is "port unreachable".

Jitter Buffer Size Changed (ji):
This event is only included here to maintain compatibility with the previous version of this package. This event is used to indicate that the gateway has made an adjustment to the depth of the jitter buffer. The syntax for requesting notification is "ji", which tells the media gateway that the controller wants notification of any jitter buffer size changes. The syntax for notification from the media gateway to the controller is "JI(####)", where the #### is a decimal number from 1 to 65536, indicating the new size of the jitter buffer in milliseconds.

Media Start (ma):
The media start event occurs on a connection when the first valid 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 is detected on a connection. If no connection is specified, the event applies to all connections for the endpoint, regardless of when the connections are created (i.e., if a connection is not specified, the event will occur when the first valid RTP packet arrives on any one of the connections on that endpoint).

Operation complete (oc):
This is the standard definition of operation complete [1].

Operation failure (of):
This is the standard definition of operation failure [1].

Packet Loss Exceeded (pl):
Packet loss rate exceeds the threshold of the specified decimal number (with a range of 1 to 100,000) of packets per 100,000 packets, where the packet loss number is indicated in parenthesis. For example, PL(10) is a drop rate of 10 in 100,000 packets. This event is requested with a parameter indicating at what packet loss rate the Call Agent wishes to be reported. If the packet loss exceeds that value, the event is reported with that same parameter. The event is only reported once when the packet loss threshold is exceeded. Once reported, a following request will re-initiate packet loss measurements and report when the threshold is exceeded again.

Quality alert (qa):
The packet loss rate or the combination of delay and jitter exceeding a quality threshold. The quality thresholds for delay, jitter and packet loss rate are provisioned values.

RTP/RTCP Timeout (rto(<timeout>,st=<start-time>)):

This event indicates that neither RTP nor RTCP packets have been received on this connection for a period of time equal to the <timeout> value (in seconds). The timeout value can be supplied as a decimal number from 1 to 65535 in the parameter when the request is made. The <timeout> parameter will be supplied in ObservedEvents when the event is reported - it then simply repeats the value used. If an RTP or RTCP packet is received before the timer expires, then the timer is reset and re-started. The event will only be generated if the timer expires without an RTP or RTCP packet arriving on the specified connection during the specified period of time. Note that if the event is requested without the <timeout> parameter, then a default timeout of 60 seconds is assumed. The <timeout> value will still be reported in ObservedEvents, even if no timeout value was indicated in the request (the default value will be indicated in that case). This is a one-shot event in that once the event occurs, a further request is required in order to re-initialize the timer.

Another optional <start-time> parameter may also be included. This is used to indicate when the timer starts. It can have one of the following values:

* "im" for immediate i.e., the timer starts as soon as the request is received. This is the default.

* "ra" to indicate that the timer should start only after an RTCP packet has been received from the other end (i.e., the timer will be initiated when the first RTCP packet is received after the request is made). Note that in the case where the other end does not support RTCP, the timer will never be initiated.

Note that either the <timeout> or <start-time> may be included in the request, but only the <timeout> value is included in the report.

An example of a request might be as follows:

```
RQNT 2001 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: 0123456789B0
R: r/rto@364823(N)(120,st=im)
```

In this case, a notify will occur if there is a period of time when no RTP or RTCP packets have been received on connection 364823 for 120 seconds.

The resulting NTFY with observed events would be as follows:

```
NTFY 3002 ds/ds1-3/6@gw-o.whatever.net MGCP 1.0
X: 0123456789B0
O: r/rto@364823(120)
```

Sampling Rate Changed (sr):
This event is only included here to maintain compatibility with the previous version of this package. This event indicates that the packetization period changed to some decimal number in milliseconds enclosed in parenthesis, as in SR(20).

Used Codec Changed (uc):
This event is only included here to maintain compatibility with the previous version of this package. This event is requested without a parameter, but when reported, the hexadecimal payload type is enclosed in parenthesis, as in UC(8), to indicate the codec was changed to PCM A-law. Codec Numbers are specified in RFC 3551 [26], or in a new definition of the audio profiles for RTP that replaces this RFC.

2.11. Resource Reservation Package

Package Name: RES
Version: 0

2.11.1. Description

The "RES" package provides local connection option support for resource reservations as well as an event to indicate reservation loss.

A number of LocalConnectionOption parameters are used in doing resource reservations: "reservation request", "reservation direction", "reservation confirmation" and "resource sharing".

Reservation Request LocalConnectionOption: The gateways can be instructed to perform a reservation on a given connection using RSVP. When a reservation is needed, the Call Agent will specify the reservation profile that should be used, which is either "controlled load" or "guaranteed service". The absence of reservation can be indicated by asking for the "best effort" service, which is the default value for this parameter.

Whether or not RSVP will be done is dependent on whether the reservation request LocalConnectionOption parameter has been included in a connection request for this connection (with either "controlled load" or "guaranteed service" indicated). If a modify connection

(MDCX) request requires a change in the reservation and the "reservation request" parameter is not included in the LocalConnectionOptions, but was included in the LocalConnectionOptions for a previous connection request for that connection, then the "reservation request" value defaults to its previously saved value for that connection. If a modify connection (MDCX) request explicitly contains a "reservation request", indicating a request for "best effort" for a connection that has an existing reservation, the existing reservation will be torn down.

Reservation Direction LocalConnectionOption:
When reservation has been requested on a connection, the gateway will examine the reservation direction LocalConnectionOption parameter to determine the direction that the reservations require and do the following:

* Start emitting RSVP "PATH" messages if the reservation direction LocalConnectionOptions parameter specified "send-only" or "send-receive".

* Start emitting RSVP "RESV" messages as soon as it receives "PATH" messages if the reservation direction parameter specified "receive-only" or "send-receive".

If an RSVP reservation is requested, but the reservation direction LocalConnectionOption parameter is missing, the reservation direction defaults to the previously saved value of the reservation direction parameter for that connection. If there was no previous reservation direction parameter for that connection, the value is deduced from the connection mode. That is:

* Start emitting RSVP "PATH" messages if the connection is in "send-only", "send-receive", "conference", "network loop back" or "network continuity test" mode (if a remote connection descriptor has been received).

* Start emitting RSVP "RESV" messages as soon as it receives "PATH" messages if the connection is in "receive-only", "send-receive", "conference", "network loop back" or "network continuity test" mode.

Reservation Confirmation LocalConnectionOption:
Another LocalConnectionOption parameter for RSVP reservations is the reservation confirmation parameter, which determines what the resource reservation pre-condition (see [1]) is for acknowledging a successful connection request:

* If the reservation confirmation parameter is set to "none", the gateway will "Ack" the connection request without waiting for reservation completion. This is the default behavior.

* If the "reservation confirmation" parameter is set to "send-only", the gateway will "Ack" when the PATH message has been sent and the corresponding RESV is received to indicate successful reservation in the send direction.

* If the "reservation confirmation" parameter is set to "receive-only", the gateway will "Ack" when reservation confirm for a reservation has been received.

* If the reservation confirmation parameter is set to "send-receive", the gateway will "Ack" only after the PATH message has been sent and the corresponding RESV has been received for send direction, and reservation confirm has been received for the receive direction.

Note that:

Values "receive-only" and "send-receive" are triggers for the gateway to request reservation confirm (RESVCONF) when it sends out the RESV.

Pre-conditions SHOULD only be added for the direction(s) for which resource reservations have been requested. If a direction is added as a pre-condition, and that direction was not requested in the resource reservation, the direction MUST simply be ignored as a pre-condition.

In this approach, resource reservation success is the pre-condition to final acknowledgement of the connection request. If the reservation fails, the connection request also fails (error code 404 - insufficient bandwidth) - as will any other part of the transaction, e.g., a notification request included as part of the connection request. A typical example of this would be a request to ring the phone and look for off-hook, included with the connection request. If the reservation fails, the phone will not ring. Similarly, if the phone is already off-hook, the command fails and there will be no resource reservation.

A provisional response SHOULD be provided if confirmation is expected to occur outside the normal retry timers and in fact a provisional response MUST be provided if the reservation confirmation parameter has value "send-receive" (without a provisional response, SDP information cannot be returned until the

final "Ack" which will not occur until the reservation is complete. This can result in a deadlock since the SDP information typically needs to be passed to the other end in order for it to initiate the RSVP PATH message in the other direction). The SDP information and connectionID MUST be included in both the provisional response and the final response. Note that in order to ensure rapid detection of a lost final response, final responses issued after provisional responses for a transaction SHALL be acknowledged, i.e., they SHALL include an empty "ResponseAck" parameter in the final response (see [1]).

If the transaction time is outside the expected bounds (time T-HIST - see the section on provisional responses in [1]), error code 406 (transaction timeout) SHOULD be returned.

Also note that if the reservation confirmation parameter is omitted, the value of the reservation confirmation parameter defaults to its previously saved value. If there is no previously saved value for the reservation confirmation parameter, or the reservation confirmation parameter has the value "none", then successful resource reservation is not a pre-condition to providing an acknowledgement to the connection request (i.e., the gateway can "Ack" right away without waiting for the reservation to complete and a provisional response will not be necessary).

Resource Sharing LocalConnectionOption:

It may be possible to share network resources across multiple connections. An example is a call-waiting scenario, where only one connection will ever be active at a time. In a 3-way calling scenario with a similar set of connections, sharing is not possible. Only the Call Agent knows what may be possible, depending on the feature that is being invoked.

In order to allow the Call Agent to indicate that sharing is possible, a resource sharing LocalConnectionOption parameter is introduced. This parameter can have one of the following values:

* A value "$" can be specified where $ refers to "this connection". This value is used when doing a create connection and indicates the intent to share resources with this connection.

* A connection ID can be specified which indicates that this is a request to share resources with the connection having this connection ID (allowing multiple connections to share resources with the connection indicated).

   * The value can be empty, which indicates a request to no longer share the resources of this connection with other connections.

In the case of a CRCX, the default value for the resource sharing local connection option is empty, and for an MDCX, the default value is its current value.

The RSVP filters will be deduced from the characteristics of the connection. The RSVP resource profiles will be deduced from the connection's bandwidth and packetization period.

Note that if RSVP is used with PacketCable Dynamic Quality of Service [35], then the parameters in NCS [36] would be used instead of the reservation direction, confirmation and reservation sharing parameters described here.

2.11.2. Parameter Encoding

The Local Connection Options for the "RES" package consist of the following:

* The resource reservation parameter, encoded as the keyword "r", followed by a colon and the value "g" (guaranteed service), "cl" (controlled load) or "be" (best effort).

* The reservation direction parameter, encoded as the keyword "r-dir" followed by a colon and the value "sendonly", "recvonly" or "sendrecv".

* The reservation confirmation parameter, encoded as the keyword "r-cnf" followed by a colon and the value "none", "sendonly", "recvonly" or "sendrecv".

* The resource sharing parameter, encoded as the keyword "r-sh" followed by a colon and either:

   * The wild-card character "$" indicating this connection, indicating future plans to share resources with this connection, or

   * A connection ID, indicating a request to share resources with the connection having the specified connection ID (and all other connections sharing resources with that connection), or

* An empty value (i.e., "r-sh:" with no value indicated), indicating a request to no longer share the resources of this connection with other connections

Note that normally local connection options that are associated with a package have the package prefix included as per the package extension rules in [1]. The local connection options in the "RES" package are exceptions. The package prefix is not included in the case of the "RES" package because it was created before the extension rules in [1] were defined.

2.11.3. Events

The following events are included as part of the resource reservation package:

Symbol	Definition	R	S Duration
re	Resource Error	C	
rl	Resource Lost	C	

Resource Error (re):

This is an indication that an error in the resource reservation occurred during the life of the connection. This event is not requested with a parameter, but is reported with a parameter (see possible values below). This event may or may not indicate the permanent loss of the reservation (i.e., any error associated with the reservation whether permanent or temporary will be reported). If requested on an endpoint (without specifying the connection ID), the request refers to all present and future connections on that endpoint. When reported, the connectionID is always supplied along with a reason for the error indicated as a parameter. One of the following possible reasons for loss MUST be included as the parameter when the event is reported:

- "resverr" is used to indicate that a ResvErr message was received.
- "patherr" is used to indicate that a PathErr message was received.
- "other"

In addition to a parameter indicating one of the reasons above, additional information on the type of error MAY be included as a second parameter in the form of a quoted string.

Example report might include:

O: res/rl@0A3F58(resverr)

or

O: res/rl@0A3F58(resverr, "some additional commentary")

Note that this event will not be reported if an error occurs while a resource reservation is initially being set up (i.e., the event was only reported as a result of an error that occurred after the reservation was set up).

Resource Lost (rl):
Loss of reservation during the life of a connection can be reported by using the "rl" event. This event is not requested with a parameter, but is reported with a parameter (see below for possible values). If requested on an endpoint (without specifying the connection ID), the request refers to all present and future connections on that endpoint.

When reported, the connectionID is always supplied along with a reason for the loss indicated as a parameter. One of the following possible reasons for loss MUST be supplied as the parameter when the event is reported:

- "resvtear" indicating that the reservation loss was indicated by ResvTear message.

- "pathtear" indicating that the reservation loss was indicated by PathTear message.

- "other"

In addition to a parameter indicating one of the reasons above, additional information on the type of error MAY be included as a second parameter in the form of a quoted string.

Example report might include:

O: res/rl@0A3F58(ResvTear)

or

O: res/rl@0A3F58(ResvTear, "some other commentary")

Note that this event will not be reported if an error occurs while a resource reservation is initially being set up (i.e., the event is only reported if the reservation was lost after it was initially set up).

2.12. Announcement Server Package

Package Name: A
Version: 1

Symbol	Definition	R	S	Duration
ann(url)	Play an Announcement		TO, C	variable
oc	Operation Complete	x		
of	Operation Failure	x		

Changes from the previous version: change to conform to standard reporting of operation failure and operation complete events.

The announcement signal is qualified by a URL name:

```
S: ann(http://scripts.example.net/all-lines-busy.au)
```

The URL name MAY be followed by a list of initial parameters, separated by commas. However, standard parameters are not included as part of this package definition (Note: use of additional parameters is optional and would result in a proprietary interface).

The gateway SHOULD support one or more standard URL schemes such as:

* file, http, ftp (RFC 1738 [28]), which indicate where the audio file is located (where to load the file from before playing the audio file on the gateway).

* RTSP URL (section 3.2 of RFC 2326 [29]), which in this case allows the media gateway to directly initiate playing of the announcement via an RTSP server.

The pre-condition for a successful response (return code of "200") is correct syntax and capability (support is available for this request). Standard MGCP return codes apply in the case of failure. Further indications of failure are provided in the operation failure event as a comment after the name of the failed event in the form of a quoted string.

If the announcement cannot be played out for a reason determined after a successful response to the request has been provided, an operation failure event will be returned. The failure MAY be explained by some commentary (in the form of a quoted string), as in:

```
O: a/of(a/ann,"file not found")
```

The "operation complete" event will be detected when the announcement is played out.

```
O: a/oc(a/ann)
```

2.13. Script Package

Package Name: Script
Version: 1

Symbol	Definition	R	S	Duration
ir(..)	Intermediate Results/Req.	x	BR	
java(url,...)	Load & Run java script		TO	variable
oc	operation complete	x		
of	operation failure	x		
perl(url,...)	Load & Run perl script		TO	variable
tcl(url,...)	Load & Run TCL script		TO	variable
vxml(url,...)	Load & Run VXML doc.		TO	variable
xml(url,...)	Load & Run XML script		TO	variable

Changes from the previous version of the package: "vxml" was added as a language type for loading and running VXML documents; change to conform with standard reporting of operation failure and operation complete events; addition of "ir" event.

The current definition defines keywords for the most common languages. More languages may be defined in later versions of this package.

The "signal" specifying the scripting language is parameterized with a URL indicating the location of the script. The URL parameter MAY be optionally followed by a comma-separated list of arguments as initial parameters to use in running the script. URL schemes may include file ftp, or http schemes with syntax according to RFC 2396 [30]. As an example:

```
S: script/vxml(ftp://ftp.example.net/credit-card.vxml,arg1,arg2,
                                                        ...,argn)
```

The argument list "arg1,arg2,...,argn" is passed to the script/document as a list of initial parameters.

The pre-condition for a successful response (return code of "200") is correct syntax and capability (support is available for this request). Standard MGCP return codes apply in the case of failure. Some further (non-application/script specific) failure indications MAY be provided in the operation failure event as a comment in the form of a quoted string following the name of the failed event.

Example

```
O: script/of(script/vxml,"file not found")
```

The script produces an output, which consists of one or several text strings, separated by commas. This provides the return-status of the script as well as return parameters (if there are any)

```
O: script/oc(script/vxml,return-status=<status>,
                   name1=value1,name2=value2,...)
```

where <status> can have one of the values "success" or "failure". This is then followed by output parameters as a comma-separated list of name-value pairs.

Intermediate Result/Request (ir(<params>)):
This provides a way for:

* The script to inform the Call Agent of intermediate results (e.g., a case where it is important because of timing concerns to inform the Call Agent prior to operation complete).

* The script to request some information from the Call Agent.

* The Call Agent to inform the script of some event or information that may be important for the operation of the script (in this case "ir" is used as a signal).

Parameters (i.e., <params>) SHOULD be a comma-separated list of name-value pairs e.g., ir(name1=value1,name2=value2,..). The Call Agent MAY include event parameters when it requests this event, in which case, the MGCP syntax requirements require that the action be specified (e.g., "R: ir(N)(nam1=value1,name2=value2,..)").

If the Call Agent requests "ir" as a signal, at least one parameter MUST be provided.

When requesting the "ir" signal, the Call Agent MUST also repeat the original script signal. This is in order to be consistent with the semantics of TO signals in MGCP (i.e., if the original "script" signal is not included, then the signal/script will be stopped). The only problem with this is that there is a possible race condition in which a request to send an "ir" signal could occur just as the script stopped. In order to avoid this confusion, the following is RECOMMENDED: when the script signal is included with an "ir" signal, include a parameter (of the script signal) to indicate that this is not a new instance of the script i.e., if there is no script executing at the present time do not start executing a new one.

The "ir" signal is only associated with an executing script. If none are running when a request for the event/signal is made, or if a new script request is not included with the request, then the "ir" signal/event will not be executed (i.e., the "ir" event with its parameters is passed to an existing script for parsing and execution and is considered opaque as far as MGCP as concerned. If no such script exists, response code "800" will be returned, indicating that the script is not executing).

The following response code is associated with this package:

Code	Text	Explanation
800	Script not Executing	Request for "ir" signal or event but no script is executing at the time the request was received.

Note that package specific error codes include the package name following the error code. For example, if error code 800 occurs in response to a request with a transaction ID of 1001, it would be sent as:

```
800 1001 /SCRIPT
```

3. IANA Considerations

The following packages and their versions have been registered with IANA as per the instructions in [1].

Package Title	Name	Version
Announcement	A	1
DTMF	D	1
Digit Map Extension	DM1	0
Media Format	FM	0
Generic	G	1
Handset	H	1
Line	L	1
RTP	R	1
Resource Reservation	RES	0
Script	SCRIPT	1
Supplementary Tones	SST	0
Signal List	SL	0
Trunk	T	1

The following extension digit map letter has been registered with IANA:

Package	Letter
DM1	P

The following Local Connections have been registered with IANA:

Field	Name
Media Format	fmtp
Reservation Confirmation	r-cnf
Reservation Direction	r-dir
Resource Sharing	r-sh

4. Security Considerations

The MGCP packages contained in this document do not require any further security considerations beyond those indicated in the base MGCP specification [1].

5. Acknowledgements

Special thanks are due to the authors of the original MGCP 1.0 specification: Mauricio Arango, Andrew Dugan, Isaac Elliott, Christian Huitema, and Scott Picket.

Thanks also to the reviewers of this document, including but not limited to: Jerry Kamitses, Sonus Networks; Dave Auerbach, Dan Wing, Cisco Systems; Ed Guy, EMC Software; Martin Wakley, Nortel Networks.

6. References

6.1. Normative References

[1] Andreasen, F. and B. Foster, "Media Gateway Control Protocol (MGCP) Version 1.0", RFC 3435, January 2003.

[2] Bellcore, "LSSGR: Switching System Generic Requirements for Call Control Using the Integrated Services Digital Network User Part (ISDNUP)", GR-317-CORE, Issue 2, December 1997.

[3] ITU-T, "Telephone User Part Signaling Procedures", ITU-T Q.724, November 1988.

[4] ANSI, "OAM&P - Terminating Test Line Access and Capabilities", T1.207-2000.

[5] Bellcore, "Notes on the Network", Special Report SR-2275, Issue 3, December 1997.

[6] Bellcore, "Call Processing" GR-505-CORE, Issue 1, December 1997.

[7] Bellcore, "LSSGR: Signaling for Analog Interfaces", GR-506-CORE, Issue 1, June 1996.

[8] ITU-T, "Technical Characteristics of Tones for the Telephone Service", ITU-T E.180, March 1998.

[9] ITU-T, "Various Tones Used in National Networks", ITU-T E.180, Supplement 2, January 1994.

[10] ITU-T, "Applications of Tones and Recorded Announcements in Telephone Services", ITU-T E.182, March 1998.

[11] Bellcore, "Call Forwarding Sub-Features FSD-01-02-1450, GR-586, Issue 1, June 2000.

[12] Bellcore, "CPE Compatibility Considerations for the Voiceband Data Transmission Interface", SR-TSV-002476, December 1992.

[13] Bellcore, "LSSGR: Visual Message Waiting Indicator Generic Requirements (FSD 01-02-2000)", GR-1401, Issue 01, June 2000.

[14] Bellcore, "LSSGR Voiceband Data Transmission Interface", Section 6.6, GR-30-CORE, Issue 02, December 1998.

[15] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[16] Bellcore, "LSSGR: Call Waiting, FSD 01-02-1201", GR-571, Issue 01, June 2000.

[17] Bellcore, "LSSGR: Verification Connections FSD 25-05-0903", GR-531-CORE, Issue 1, June 2000.

[18] Bellcore, " LSSGR: CLASS Feature: Calling Identity Delivery on Call Waiting, FSD 01-02-1090, GR-575, Issue 01, June 2000.

[19] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[20] Bellcore, "Class Feature: Screen Editing (FSD 30-28-0000)", GR-220, Issue 2, April 2002.

[21] ITU-T, "Procedure for document facsimile transmission in the general switched telephone network", ITU-T T.30, April 1999.

[22] ITU-T, "300 bits per second duplex modem standardized for use in the general switched telephone network", ITU-T V.21, November 1988.

[23] Telcordia Technologies, "Telcordia Technologies Specification of Signaling System Number 7", GR-246-CORE, Issue 7, December 2002.

[24] Telcordia Technologies, "LSSGR: CLASS Feature: Calling Name Delivery Generic Requirements (FSD 01-02-1070)", GR-1188, Issue 02, December 2000.

[25] Telcordia Technologies, "LSSGR: CLASS Feature: Calling Number Delivery (FSD 01-02-1051)", GR-31, Issue 01, June 2000.

[26] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 3551, July 2003.

[27] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[28] Berners-Lee, T., Masinter, L. and M. McCahill, Eds., "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[29] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[30] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[31] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[32] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J.C., Vega-Garcia, A. and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[33] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[34] Foster, B., "MGCP CAS Packages", RFC 3064, February 2001.

[35] PacketCableTM, Dynamic Quality if Service Specification, http://www.packetcable.com/downloads/specs/PKT-SP-DQOS-I07-030815.pdf

[36] PacketCableTM Network-Based Call Signaling Protocol http://www.packetcable.com/downloads/specs/PKT-SP-EC-MGCP-I08-030728.pdf

[37] Groves, C., Pantaleo, M., Anderson, T. and T. Taylor, Eds., "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[38] Arango, M., Dugan, A., Elliott, I., Huitema, C. and S. Pickett, "Media Gateway Control Protocol (MGCP) Version 1.0", RFC 2705, October 1999.

7. Authors' Addresses

Bill Foster
Cisco Systems

Phone: +1 250 758 9418
EMail: bfoster@cisco.com

Flemming Andreasen
Cisco Systems
Edison, NJ 08837

EMail: fandreas@cisco.com

8. Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.