Internet Engineering Task Force (IETF)
Request for Comments: 6230
Category: Standards Track
ISSN: 2070-1721

C. Boulton
NS-Technologies
T. Melanchuk
Rainwillow
S. McGlashan
Hewlett-Packard

May 2011

Media Control Channel Framework

Abstract

This document describes a framework and protocol for application deployment where the application programming logic and media processing are distributed. This implies that application programming logic can seamlessly gain access to appropriate resources that are not co-located on the same physical network entity. The framework uses the Session Initiation Protocol (SIP) to establish an application-level control mechanism between application servers and associated external servers such as media servers.

The motivation for the creation of this framework is to provide an interface suitable to meet the requirements of a centralized conference system, where the conference system can be distributed, as defined by the XCON working group in the IETF. It is not, however, limited to this scope.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6230.

Table of Contents

1. Introduction

Real-time media applications are often developed using an architecture where the application logic and media processing activities are distributed. Commonly, the application logic runs on "application servers", but the processing runs on external servers, such as "media servers". This document focuses on the framework and protocol between the application server and external processing server. The motivation for this framework comes from a set of requirements for Media Server Control, which can be found in "Media Server Control Protocol Requirements" [RFC5167]. While the Framework is not specific to media server control, it is the primary driver and use case for this work. It is intended that the framework contained in this document be able to be used for a variety of device control scenarios (for example, conference control).

This document does not define a particular SIP extension for the direct control of external components. Rather, other documents, known as "Control Packages", extend the Control Framework described by this document. Section 8 provides a comprehensive set of guidelines for creating such Control Packages.

Current IETF device control protocols, such as Megaco [RFC5125], while excellent for controlling media gateways that bridge separate networks, are troublesome for supporting media-rich applications in SIP networks. This is because Megaco duplicates many of the functions inherent in SIP. Rather than using a single protocol for session establishment and application media processing, application developers need to translate between two separate mechanisms. Moreover, the model provided by the framework presented here, using SIP, better matches the application programming model than does Megaco.

SIP [RFC3261] provides the ideal rendezvous mechanism for establishing and maintaining control connections to external server components. The control connections can then be used to exchange explicit command/response interactions that allow for media control and associated command response results.

2. Conventions and Terminology

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, [RFC2119], as scoped to those conformance targets.

The following additional terms are defined for use in this document:

User Agent Client (UAC): As specified in [RFC3261].

User Agent Server (UAS): As specified in [RFC3261].

B2BUA: A B2BUA is a Back-to-Back SIP User Agent.

Control Server: A Control Server is an entity that performs a service, such as media processing, on behalf of a Control Client. For example, a media server offers mixing, announcement, tone detection and generation, and play and record services. The Control Server has a direct Real-Time Transport Protocol (RTP) [RFC3550] relationship with the source or sink of the media flow. In this document, we often refer to the Control Server simply as "the Server".

Control Client: A Control Client is an entity that requests processing from a Control Server. Note that the Control Client might not have any processing capabilities whatsoever. For example, the Control Client may be an application server (B2BUA) or other endpoint requesting manipulation of a third party's media stream that terminates on a media server acting in the role of a Control Server. In this document, we often refer to the Control Client simply as "the Client".

Control Channel: A Control Channel is a reliable connection between a Client and Server that is used to exchange Framework messages. The term "Connection" is used synonymously within this document.

Framework Message: A Framework message is a message on a Control Channel that has a type corresponding to one of the Methods defined in this document. A Framework message is often referred to by its method, such as a "CONTROL message".

Method: A Method is the type of a Framework message. Four Methods are defined in this document: SYNC, CONTROL, REPORT, and K-ALIVE.

Control Command: A Control Command is an application-level request from a Client to a Server. Control Commands are carried in the body of CONTROL messages. Control Commands are defined in separate specifications known as "Control Packages".

Framework Transaction: A Framework Transaction is defined as a sequence composed of a Control Framework message originated by either a Control Client or Control Server and responded to with a Control Framework response code message. Note that the Control Framework has no "provisional" responses. A Control Framework

transaction is referenced throughout the document as a 'Transaction-Timeout'.

Transaction-Timeout: The maximum allowed time between a Control Client or Server issuing a Framework message and it arriving at the destination. The value for 'Transaction-Timeout' is 10 seconds.

3. Overview

This document details mechanisms for establishing, using, and terminating a reliable transport connection channel using SIP and the Session Description Protocol offer/answer [RFC3264] exchange. The established connection is then used for controlling an external server. The following text provides a non-normative overview of the mechanisms used. Detailed, normative guidelines are provided later in the document.

Control Channels are negotiated using standard SIP mechanisms that would be used in a similar manner to creating a SIP multimedia session. Figure 1 illustrates a simplified view of the mechanism. It highlights a separation of the SIP signaling traffic and the associated Control Channel that is established as a result of the SIP interactions.

Initial analysis into the Control Framework, as documented in [MSCL-THOUGHTS], established the following. One might ask, "If all we are doing is establishing a TCP connection to control the media server, why do we need SIP?" This is a reasonable question. The key is that we use SIP for media session establishment. If we are using SIP for media session establishment, then we need to ensure the URI used for session establishment resolves to the same node as the node for session control. Using the SIP routing mechanism, and having the server initiate the TCP connection back, ensures this works. For example, the URI sip:myserver.example.com may resolve to sip: server21.farm12.northeast.example.net, whereas the URI http://myserver.example.com may resolve to http://server41.httpfarm.central.example.net. That is, the host part is not necessarily unambiguous.

The use of SIP to negotiate the Control Channel provides many inherent capabilities, which include:

- Service location - Use SIP Proxies and Back-to-Back User Agents for locating Control Servers.

- Security mechanisms - Leverage established security mechanisms such as Transport Layer Security (TLS) and Client Authentication.

- Connection maintenance - The ability to re-negotiate a connection, ensure it is active, and so forth.

- Application agnostic - Generic protocol allows for easy extension.

As mentioned in the previous list, one of the main benefits of using SIP as the session control protocol is the "Service Location" facilities provided. This applies both at a routing level, where [RFC3263] provides the physical location of devices, and at the service level, using Caller Preferences [RFC3840] and Callee Capabilities [RFC3841]. The ability to select a Control Server based on service-level capabilities is extremely powerful when considering a distributed, clustered architecture containing varying services (for example, voice, video, IM). More detail on locating Control Server resources using these techniques is outlined in Section 4.1 of this document.

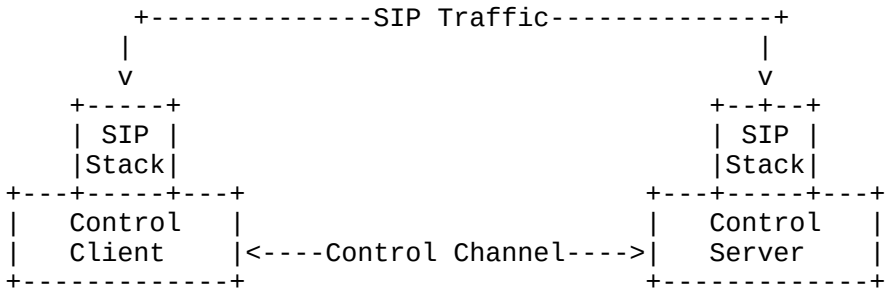


Figure 1: Basic Architecture

The example from Figure 1 conveys a 1:1 connection between the Control Client and the Control Server. It is possible, if required, for the client to request multiple Control Channels using separate SIP INVITE dialogs between the Control Client and the Control Server entities. Any of the connections created between the two entities can then be used for Server control interactions. The control connections are orthogonal to any given media session. Specific media session information is incorporated in control interaction commands, which themselves are defined in external packages, using the XML schema defined in Appendix A. The ability to have multiple Control Channels allows for stronger redundancy and the ability to manage high volumes of traffic in busy systems.

Consider the following simple example for session establishment between a Client and a Server. (Note: Some lines in the examples are removed for clarity and brevity.) Note that the roles discussed are logical and can change during a session, if the Control Package allows.

The Client constructs and sends a standard SIP INVITE request, as defined in [RFC3261], to the external Server. The Session Description Protocol (SDP) payload includes the required information for Control Channel negotiation and is the primary mechanism for conveying support for this specification. The application/cfw MIME type is defined in this document to convey the appropriate SDP format for compliance to this specification. The Connection-Oriented Media (COMEDIA) [RFC4145] specification for setting up and maintaining reliable connections is used as part of the negotiation mechanism (more detail available in later sections). The Client also includes the 'cfw-id' SDP attribute, as defined in this specification, which is a unique identifier used to correlate the underlying Media Control Channel with the offer/answer exchange.

Client Sends to External Server:

```
INVITE sip:External-Server@example.com SIP/2.0
To: <sip:External-Server@example.com>
From: <sip:Client@example.com>;tag=64823746
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK72d
Call-ID: 7823987HJHG6
Max-Forwards: 70
CSeq: 1 INVITE
Contact: <sip:Client@clientmachine.example.com>
Content-Type: application/sdp
Content-Length: [..]

v=0
o=originator 2890844526 2890842808 IN IP4 controller.example.com
s=-
c=IN IP4 controller.example.com
m=application 49153 TCP cfw
a=setup:active
a=connection:new
a=cfw-id:H839quwhjdhegvdga
```

On receiving the INVITE request, an external Server supporting this mechanism generates a 200 OK response containing appropriate SDP and formatted using the application/cfw MIME type specified in this document. The Server inserts its own unique 'cfw-id' SDP attribute, which differs from the one received in the INVITE (offer).

External Server Sends to Client:

```
SIP/2.0 200 OK
To: <sip:External-Server@example.com>;tag=28943879
From: <sip:Client@example.com>;tag=64823746
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK72d;received=192.0.2.4
```

```
Call-ID: 7823987HJHG6
CSeq: 1 INVITE
Contact: <sip:External-Server@servermachine.example.com>
Content-Type: application/sdp
Content-Length: [..]

v=0
o=responder 2890844526 2890842808 IN IP4 server.example.com
s=-
c=IN IP4 mserver.example.com
m=application 7563 TCP cfw
a=setup:passive
a=connection:new
a=cfw-id:U8dh7UHDushsdu32uha
```

The Control Client receives the SIP 200 OK response and extracts the relevant information (also sending a SIP ACK). It creates an outgoing (as specified by the SDP 'setup' attribute of 'active') TCP connection to the Control Server. The connection address (taken from 'c=') and port (taken from 'm=') are used to identify the remote port in the new connection.

Once established, the newly created connection can be used to exchange requests and responses as defined in this document. If required, after the Control Channel has been set up, media sessions can be established using standard SIP Third Party Call Control (3PCC) [RFC3725].

Figure 2 provides a simplified example where the framework is used to control a User Agent's RTP session.

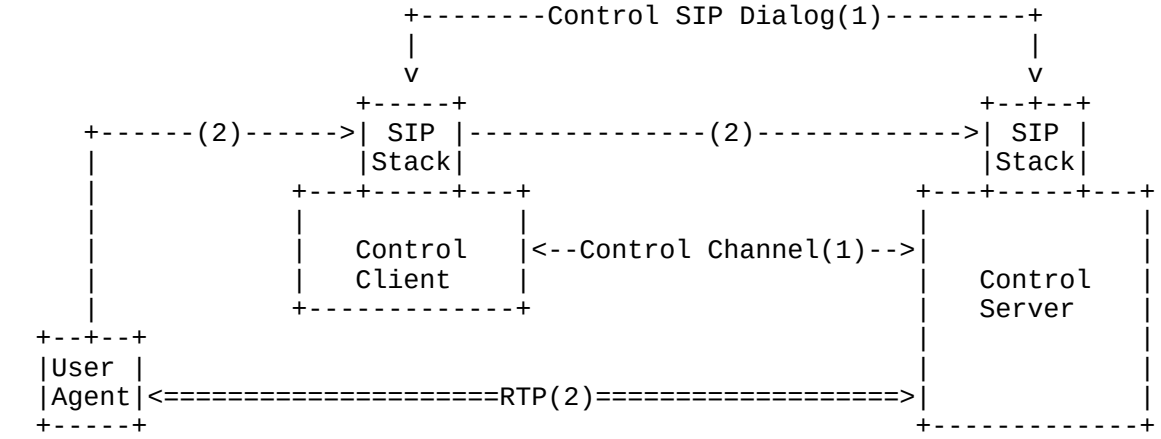


Figure 2: Participant Architecture

The link (1) represents the SIP INVITE dialog usage and dedicated Control Channel previously described in this overview section. The link (2) from Figure 2 represents the User Agent SIP INVITE dialog usage interactions and associated media flow. A User Agent creates a SIP INVITE dialog usage with the Control Client entity. The Control Client entity then creates a SIP INVITE dialog usage to the Control Server, using B2BUA type functionality. Using the interaction illustrated by (2), the Control Client negotiates media capabilities with the Control Server, on behalf of the User Agent, using SIP 3PCC. [RFC3725].

4. Control Channel Setup

This section describes the setup, using SIP, of the dedicated Control Channel. Once the Control Channel has been established, commands can be exchanged (as discussed in Section 6).

4.1. Control Client SIP UAC Behavior

When a UAC wishes to establish a Control Channel, it MUST construct and transmit a new SIP INVITE request for Control Channel setup. The UAC MUST construct the INVITE request as defined in [RFC3261].

If a reliable response is received (as defined in [RFC3261] and [RFC3262]), the mechanisms defined in this document are applicable to the newly created SIP INVITE dialog usage.

The UAC SHOULD include a valid session description (an 'offer' as defined in [RFC3264]) in an INVITE request using the Session Description Protocol defined in [RFC4566] but MAY choose an offer-less INVITE as per [RFC3261]. The SDP SHOULD be formatted in accordance with the steps below and using the MIME type application/cfw, which is registered in Section 13. The following information defines the composition of specific elements of the SDP payload the offerer MUST adhere to when used in a SIP-based offer/answer exchange using SDP and the application/cfw MIME type. The SDP being constructed MUST contain only a single occurrence of a Control Channel definition outlined in this specification but can contain other media lines if required.

The Connection Data line in the SDP payload is constructed as specified in [RFC4566]:

c=<nettype> <addrtype> <connection-address>

The first sub-field, <nettype>, MUST equal the value "IN". The second sub-field, <addrtype>, MUST equal either "IP4" or "IP6". The third sub-field for Connection Data is <connection-address>. This

supplies a representation of the SDP originator's address, for example, DNS/IP representation. The address is the address used for connections.

Example:

c=IN IP4 controller.example.com

The SDP MUST contain a corresponding Media Description entry:

m=<media> <port> <proto> <fmt>

The first "sub-field", <media>, MUST equal the value "application". The second sub-field, <port>, MUST represent a port on which the constructing client can receive an incoming connection if required. The port is used in combination with the address specified in the Connection Data line defined previously to supply connection details. If the entity constructing the SDP can't receive incoming connections, it must still enter a valid port entry. The use of the port value '0' has the same meaning as defined in a SIP offer/answer exchange [RFC3264]. The Control Framework has a default port defined in Section 13.5. This value is default, although a client is free to choose explicit port numbers. However, SDP SHOULD use the default port number, unless local policy prohibits its use. Using the default port number allows network administrators to manage firewall policy for Control Framework interactions. The third sub-field, <proto>, compliant to this specification, MUST support the values "TCP" and "TCP/TLS". Implementations MUST support TLS as a transport-level security mechanism for the Control Channel, although use of TLS in specific deployments is optional. Control Framework implementations MUST support TCP as a transport protocol. When an entity identifies a transport value but is not willing to establish the session, it MUST respond using the appropriate SIP mechanism. The <fmt> sub-field MUST contain the value "cfw".

The SDP MUST also contain a number of SDP media attributes (a=) that are specifically defined in the COMEDIA [RFC4145] specification. The attributes provide connection negotiation and maintenance parameters. It is RECOMMENDED that a Controlling UAC initiate a connection to an external Server but that an external Server MAY negotiate and initiate a connection using COMEDIA, if network topology prohibits initiating connections in a certain direction. An example of the COMEDIA attributes is:

```
a=setup:active
a=connection:new
```

This example demonstrates a new connection that will be initiated from the owner of the SDP payload. The connection details are contained in the SDP answer received from the UAS. A full example of an SDP payload compliant to this specification can be viewed in Section 3. Once the SDP has been constructed along with the remainder of the SIP INVITE request (as defined in [RFC3261]), it can be sent to the appropriate location. The SIP INVITE dialog usage and appropriate control connection is then established.

A SIP UAC constructing an offer MUST include the 'cfw-id' SDP attribute as defined in Section 9.2. The 'cfw-id' attribute indicates an identifier that can be used within the Control Channel to correlate the Control Channel with this SIP INVITE dialog usage. The 'cfw-id' attribute MUST be unique in the context of the interaction between the UAC and UAS and MUST NOT clash with instances of the 'cfw-id' used in other SIP offer/answer exchanges. The value chosen for the 'cfw-id' attribute MUST be used for the entire duration of the associated SIP INVITE dialog usage and not be changed during updates to the offer/answer exchange. This applies specifically to the 'connection' attribute as defined in [RFC4145]. If a SIP UAC wants to change some other parts of the SDP but reuse the already established connection, it uses the value of 'existing' in the 'connection' attribute (for example, a=connection:existing). If it has noted that a connection has failed and wants to re-establish the connection, it uses the value of 'new' in the 'connection' attribute (for example, a=connection:new). Throughout this, the connection identifier specified in the 'cfw-id' SDP parameter MUST NOT change. One is simply negotiating the underlying TCP connection between endpoints but always using the same Control Framework session, which is 1:1 for the lifetime of the SIP INVITE dialog usage.

A non-2xx-class final SIP response (3xx, 4xx, 5xx, and 6xx) received for the INVITE request indicates that no SIP INVITE dialog usage has been created and is treated as specified by SIP [RFC3261]. Specifically, support of this specification is negotiated through the presence of the media type defined in this specification. The receipt of a SIP error response such as "488" indicates that the offer contained in a request is not acceptable. The inclusion of the media line associated with this specification in such a rejected offer indicates to the client generating the offer that this could be due to the receiving client not supporting this specification. The client generating the offer MUST act as it would normally on receiving this response, as per [RFC3261]. Media streams can also be rejected by setting the port to "0" in the "m=" line of the session description, as defined in [RFC3264]. A client using this specification MUST be prepared to receive an answer where the "m=" line it inserted for using the Control Framework has been set to "0".

In this situation, the client will act as it would for any other media type with a port set to "0".

4.2. Control Server SIP UAS Behavior

On receiving a SIP INVITE request, an external Server (SIP UAS) inspects the message for indications of support for the mechanisms defined in this specification. This is achieved through inspection of the session description of the offer message and identifying support for the application/cfw MIME type in the SDP. If the SIP UAS wishes to construct a reliable response that conveys support for the extension, it MUST follow the mechanisms defined in [RFC3261]. If support is conveyed in a reliable SIP provisional response, the mechanisms in [RFC3262] MUST also be used. It should be noted that the SDP offer is not restricted to the initial INVITE request and MAY appear in any series of messages that are compliant to [RFC3261], [RFC3262], [RFC3311], and [RFC3264].

When constructing an answer, the SDP payload MUST be constructed using the semantic (connection, media, and attribute) defined in Section 4.1 using valid local settings and also with full compliance to the COMEDIA [RFC4145] specification. For example, the SDP attributes included in the answer constructed for the example offer provided in Section 4.1 would look as follows:

```
a=setup:passive
a=connection:new
```

A client constructing an answer MUST include the 'cfw-id' SDP attribute as defined in Section 9.2. This attribute MUST be unique in the context of the interaction between the UAC and UAS and MUST NOT clash with instances of the 'cfw-id' used in other SIP offer/answer exchanges. The 'cfw-id' MUST be different from the 'cfw-id' value received in the offer as it is used to uniquely identify and distinguish between multiple endpoints that generate SDP answers. The value chosen for the 'cfw-id' attribute MUST be used for the entire duration of the associated SIP INVITE dialog usage and not be changed during updates to the offer/answer exchange.

Once the SDP answer has been constructed, it is sent using standard SIP mechanisms. Depending on the contents of the SDP payloads that were negotiated using the offer/answer exchange, a reliable connection will be established between the Controlling UAC and External Server UAS entities. The newly established connection is now available to exchange Control Command primitives. The state of the SIP INVITE dialog usage and the associated Control Channel are now implicitly linked. If either party wishes to terminate a Control Channel, it simply issues a SIP termination request (for example, a

SIP BYE request or appropriate response in an early SIP INVITE dialog usage). The Control Channel therefore lives for the duration of the SIP INVITE dialog usage.

A UAS receiving a SIP OPTIONS request MUST respond appropriately as defined in [RFC3261]. The UAS MUST include the media types supported in the SIP 200 OK response in a SIP 'Accept' header to indicate the valid media types.

5. Establishing Media Streams - Control Client SIP UAC Behavior

It is intended that the Control Framework will be used within a variety of architectures for a wide range of functions. One of the primary functions will be the use of the Control Channel to apply multiple specific Control Package commands to media sessions established by SIP INVITE dialogs (media dialogs) with a given remote server. For example, the Control Server might send a command to generate audio media (such as an announcement) on an RTP stream between a User Agent and a media server.

SIP INVITE dialogs used to establish media sessions (see Figure 2) on behalf of User Agents MAY contain more than one Media Description (as defined by "m=" in the SDP). The Control Client MUST include a media label attribute, as defined in [RFC4574], for each "m=" definition received that is to be directed to an entity using the Control Framework. This allows the Control Client to later explicitly direct commands on the Control Channel at a specific media line (m=).

This framework identifies the referencing of such associated media dialogs as extremely important. A connection reference attribute has been specified that can optionally be imported into any Control Package. It is intended that this will reduce the repetitive specifying of dialog reference language. The schema can be found in Appendix A.1.

Similarly, the ability to identify and apply commands to a group of associated media dialogs (multiparty) is also identified as a common structure that could be defined and reused, for example, playing a prompt to all participants in a Conference. The schema for such operations can also be found in Appendix A.1.

Support for both the common attributes described here is specified as part of each Control Package definition, as detailed in Section 8.

6. Control Framework Interactions

In this document, the use of the COMEDIA specification allows for a Control Channel to be set up in either direction as a result of a SIP INVITE transaction. SIP provides a flexible negotiation mechanism to establish the Control Channel, but there needs to be a mechanism within the Control Channel to correlate it with the SIP INVITE dialog usage implemented for its establishment. A Control Client receiving an incoming connection (whether it be acting in the role of UAC or UAS) has no way of identifying the associated SIP INVITE dialog usage as it could be simply listening for all incoming connections on a specific port. The following steps, which implementations MUST support, allow a connecting UA (that is, the UA with the active role in COMEDIA) to identify the associated SIP INVITE dialog usage that triggered the connection. Unless there is an alternative dialog association mechanism used, the UAs MUST carry out these steps before any other signaling on the newly created Control Channel.

- Once the connection has been established, the UA acting in the active role (active UA) to initiate the connection MUST send a Control Framework SYNC request. The SYNC request MUST be constructed as defined in Section 9.1 and MUST contain the 'Dialog-ID' message header.

- The 'Dialog-ID' message header is populated with the value of the local 'cfw-id' media-level attribute that was inserted by the same client in the SDP offer/answer exchange to establish the Control Channel. This allows for a correlation between the Control Channel and its associated SIP INVITE dialog usage.

- On creating the SYNC request, the active UA MUST follow the procedures outlined in Section 6.3.3. This provides details of connection keep-alive messages.

- On creating the SYNC request, the active UA MUST also follow the procedures outlined in Section 6.3.4.2. This provides details of the negotiation mechanism used to determine the Protocol Data Units (PDUs) that can be exchanged on the established Control Channel connection.

- The UA in the active role for the connection creation MUST then send the SYNC request. If the UA in the active role for the connection creation is a SIP UAS and has generated its SDP response in a 2xx-class SIP response, it MUST wait for an incoming SIP ACK message before issuing the SYNC. If the UA in the active role for the connection creation is a SIP UAS and has generated its SDP response in a reliable 1XX class SIP response, it MUST wait for an incoming SIP PRACK message before issuing the SYNC.

If the UA in the active role for the connection creation is a SIP UAC, it MUST send the SYNC message immediately on establishment of the Control Channel. It MUST then wait for a period of at least 2*'Transaction-Timeout' to receive a response. It MAY choose a longer time to wait, but it MUST NOT be shorter than 'Transaction-Timeout'. In general, a Control Framework transaction MUST complete within 20 (2*'Transaction-Timeout') seconds and is referenced throughout the document as 'Transaction-Timeout'.

- If no response is received for the SYNC message, a timeout occurs and the Control Channel is terminated along with the associated SIP INVITE dialog usage. The active UA MUST issue a BYE request to terminate the SIP INVITE dialog usage.

- If the active UA receives a 481 response from the passive UA, this means the SYNC request was received, but the associated SIP INVITE dialog usage specified in the SYNC message does not exist. The active client MUST terminate the Control Channel. The active UA MUST issue a SIP BYE request to terminate the SIP INVITE dialog usage.

- All other error responses received for the SYNC request are treated as detailed in this specification and also result in the termination of the Control Channel and the associated SIP INVITE dialog usage. The active UA MUST issue a BYE request to terminate the SIP INVITE dialog usage.

- The receipt of a 200 response to a SYNC message implies that the SIP INVITE dialog usage and control connection have been successfully correlated. The Control Channel can now be used for further interactions.

SYNC messages can be sent at any point while the Control Channel is open from either side, once the initial exchange is complete. If present, the contents of the 'Keep-Alive' and 'Dialog-ID' headers MUST NOT change. New values of the 'Keep-Alive' and 'Dialog-ID' headers have no relevance as they are negotiated for the lifetime of the Media Control Channel Framework session.

Once a successful Control Channel has been established, as defined in Sections 4.1 and 4.2, and the connection has been correlated, as described in previous paragraphs, the two entities are now in a position to exchange Control Framework messages. The following sub-sections specify the general behavior for constructing Control Framework requests and responses. Section 6.3 specifies the core Control Framework methods and their transaction processing.

6.1. General Behavior for Constructing Requests

An entity acting as a Control Client that constructs and sends requests on a Control Channel MUST adhere to the syntax defined in Section 9. Note that either entity can act as a Control Client depending on individual package requirements. Control Commands MUST also adhere to the syntax defined by the Control Packages negotiated in Sections 4.1 and 4.2 of this document. A Control Client MUST create a unique transaction and associated identifier for insertion in the request. The transaction identifier is then included in the first line of a Control Framework message along with the method type, as defined in the ABNF in Section 9. The first line starts with the "CFW" token for the purpose of easily extracting the transaction identifier. The transaction identifier MUST be unique in the context of the interaction between the Control Client and Control Server. This unique property helps avoid clashes when multiple client entities could be creating transactions to be carried out on a single receiving server. All required, mandatory, and optional Control Framework headers are then inserted into the request with appropriate values (see relevant individual header information for explicit detail). A 'Control-Package' header MUST also be inserted with the value indicating the Control Package to which this specific request applies. Multiple packages can be negotiated per Control Channel using the SYNC message discussed in Section 6.3.4.2.

Any Framework message that contains an associated payload MUST also include the 'Content-Type' and 'Content-Length' message headers, which indicate the MIME type of the payload specified by the individual Control Framework packages and the size of the message body represented as a whole decimal number of octets, respectively. If no associated payload is to be added to the message, the 'Content-Length' header MUST have a value of '0'.

A Server receiving a Framework message request MUST respond with an appropriate response (as defined in Section 6.2). Control Clients MUST wait for a minimum of 2*'Transaction-Timeout' for a response before considering the transaction a failure and tidying state appropriately depending on the extension package being used.

6.2. General Behavior for Constructing Responses

An entity acting as a Control Server, on receiving a request, MUST generate a response within the 'Transaction-Timeout', as measured from the Control Client. The response MUST conform to the ABNF defined in Section 9. The first line of the response MUST contain the transaction identifier used in the first line of the request, as

defined in Section 6.1. Responses MUST NOT include the 'Status' or 'Timeout' message headers, and these MUST be ignored if received by a Client in a response.

A Control Server MUST include a status code in the first line of the response. If there is no error, the Server responds with a 200 Control Framework status code, as defined in Section 7.1. The 200 response MAY include message bodies. If the response contains a payload, the message MUST include the 'Content-Length' and 'Content-Type' headers. When the Control Client receives a 2xx-class response, the Control Command transaction is complete.

If the Control Server receives a request, like CONTROL, that the Server understands, but the Server knows processing the command will exceed the 'Transaction-Timeout', then the Server MUST respond with a 202 status code in the first line of the response. Following the initial response, the server will send one or more REPORT messages as described in Section 6.3.2. A Control Package MUST explicitly define the circumstances under which the server sends 200 and 202 messages.

If a Control Server encounters problems with a Control Framework request (like REPORT or CONTROL), an appropriate error code MUST be used in the response, as listed in Section 7. The generation of a non-2xx-class response code to a Control Framework request (like CONTROL or REPORT) will indicate failure of the transaction, and all associated transaction state and resources MUST be terminated. The response code may provide an explicit indication of why the transaction failed, which might result in a re-submission of the request depending on the extension package being used.

6.3. Transaction Processing

The Control Framework defines four types of requests (methods): CONTROL, REPORT, K-ALIVE, and SYNC. Implementations MUST support sending and receiving these four methods.

The following sub-sections specify each Control Framework method and its associated transaction processing.

6.3.1. CONTROL Transactions

A CONTROL message is used by the Control Client to pass control-related information to a Control Server. It is also used as the event-reporting mechanism in the Control Framework. Reporting events is simply another usage of the CONTROL message, which is permitted to be sent in either direction between two participants in a session, carrying the appropriate payload for an event. The message is constructed in the same way as any standard Control Framework

message, as discussed in Section 6.1 and defined in Section 9. A CONTROL message MAY contain a message body. The explicit Control Command(s) of the message payload contained in a CONTROL message are specified in separate Control Package specifications. Separate Control Package specifications MUST conform to the format defined in Section 8.4. A CONTROL message containing a payload MUST include a 'Content-Type' header. The payload MUST be one of the payload types defined by the Control Package. Individual packages MAY allow a CONTROL message that does not contain a payload. This could in fact be a valid message exchange within a specific package; if it's not, an appropriate package-level error message MUST be generated.

6.3.2. REPORT Transactions

A 'REPORT' message is used by a Control Server when processing of a CONTROL command extends beyond the 'Transaction-Timeout', as measured from the Client. In this case, the Server returns a 202 response. The Server returns status updates and the final results of the command in subsequent REPORT messages.

All REPORT messages MUST contain the same transaction ID in the request start line that was present in the original CONTROL transaction. This correlates extended transactions with the original CONTROL transaction. A REPORT message containing a payload MUST include the 'Content-Type' and 'Content-Length' headers indicating the payload MIME type [RFC2045] defined by the Control Package and the length of the payload, respectively.

6.3.2.1. Reporting the Status of Extended Transactions

On receiving a CONTROL message, a Control Server MUST respond within 'Transaction-Timeout' with a status code for the request, as specified in Section 6.2. If the processing of the command completes within that time, a 200 response code MUST be sent. If the command does not complete within that time, the response code 202 MUST be sent indicating that the requested command is still being processed and the CONTROL transaction is being extended. The REPORT method is then used to update and terminate the status of the extended transaction. The Control Server should not wait until the last possible opportunity to make the decision of issuing a 202 response code and should ensure that it has plenty of time for the response to arrive at the Control Client. If it does not have time, transactions will be terminated (timed out) at the Control Client before completion.

A Control Server issuing a 202 response MUST ensure the message contains a 'Timeout' message header. This header MUST have a value in seconds that is the amount of time the recipient of the 202 message MUST wait before assuming that there has been a problem and terminating the extended transaction and associated state.

The initial REPORT message MUST contain a 'Seq' (Sequence) message header with a value equal to '1'. Note: the 'Seq' numbers at both Control Client and Control Server for Framework messages are independent.

All REPORT messages for an extended CONTROL transaction MUST contain a 'Timeout' message header. This header will contain a value in seconds that is the amount of time the recipient of the REPORT message MUST wait before assuming that there has been a problem and terminating the extended transaction and associated state. On receiving a REPORT message with a 'Status' header of 'update', the Control Client MUST reset the timer for the associated extended CONTROL transaction to the indicated timeout period. If the timeout period approaches and no intended REPORT messages have been generated, the entity acting as a Control Framework UAS for the interaction MUST generate a REPORT message containing, as defined in this paragraph, a 'Status' header of 'update' with no associated payload. Such a message acts as a timeout refresh and in no way impacts the extended transaction because no message body or semantics are permitted. It is RECOMMENDED that a minimum value of 10 and a maximum value of 15 seconds be used for the value of the 'Timeout' message header. It is also RECOMMENDED that a Control Server refresh the timeout period of the CONTROL transaction at an interval that is not too close to the expiry time. A value of 80% of the timeout period could be used. For example, if the timeout period is 10 seconds, the Server would refresh the transaction after 8 seconds.

Subsequent REPORT messages that provide additional information relating to the extended CONTROL transaction MUST also include and increment by 1 the 'Seq' header value. A REPORT message received that has not been incremented by 1 MUST be responded to with a 406 response and the extended transaction MUST be considered terminated. On receiving a 406 response, the extended transaction MUST be terminated. REPORT messages MUST also include a 'Status' header with a value of 'update'. These REPORT messages sent to update the extended CONTROL transaction status MAY contain a message body, as defined by individual Control Packages and specified in Section 8.5. A REPORT message sent updating the extended transaction also acts as a timeout refresh, as described earlier in this section. This will result in a transaction timeout period at the initiator of the original CONTROL request being reset to the interval contained in the 'Timeout' message header.

When all processing for an extended CONTROL transaction has taken place, the entity acting as a Control Server MUST send a terminating REPORT message. The terminating REPORT message MUST increment the value in the 'Seq' message header by the value of '1' from the previous REPORT message. It MUST also include a 'Status' header with a value of 'terminate' and MAY contain a message body. It MUST also contain a 'Timeout' message header with a valid value. The inclusion of the 'Timeout' header is for consistency, and its value is ignored. A Control Framework UAC can then clean up any pending state associated with the original CONTROL transaction.

6.3.3. K-ALIVE Transactions

The protocol defined in this document may be used in various network architectures. This includes a wide range of deployments where the clients could be co-located in a secured, private domain, or spread across disparate domains that require traversal of devices such as Network Address Translators (NATs) and firewalls. A keep-alive mechanism enables the Control Channel to be kept active during times of inactivity. This is because many firewalls have a timeout period after which connections are closed. This mechanism also provides the ability for application-level failure detection. It should be noted that the following procedures apply only to the Control Channel being created. For details relating to the SIP keep-alive mechanism, implementers should seek guidance from SIP Outbound [RFC5626].

The following keep-alive procedures MUST be implemented. Specific deployments MAY choose not to use the keep-alive mechanism if both entities are in a co-located domain. Note that choosing not to use the keep-alive mechanism defined in this section, even when in a co-located architecture, will reduce the ability to detect application-level errors, especially during long periods of inactivity.

Once the SIP INVITE dialog usage has been established and the underlying Control Channel has been set up, including the initial correlation handshake using SYNC as discussed in Section 6, both entities acting in the active and passive roles, as defined in COMEDIA [RFC4145], MUST start a keep-alive timer equal to the value negotiated during the Control Channel SYNC request/response exchange. This is the value from the 'Keep-Alive' header in seconds.

6.3.3.1. Behavior for an Entity in an Active Role

When in an active role, a K-ALIVE message MUST be generated before the local keep-alive timer fires. An active entity is free to send the K-ALIVE message whenever it chooses. It is RECOMMENDED for the entity to issue a K-ALIVE message after 80% of the local keep-alive timer. On receiving a 200 OK Control Framework message for the

K-ALIVE request, the active entity MUST reset the local keep-alive timer. If no 200 OK response is received to the K-ALIVE message, or a transport-level problem is detected by some other means, before the local keep-alive timer fires, the active entity MAY use COMEDIA re-negotiation procedures to recover the connection. Otherwise, the active entity MUST tear down the SIP INVITE dialog and recover the associated Control Channel resources.

6.3.3.2. Behavior for an Entity in a Passive Role

When acting as a passive entity, a K-ALIVE message must be received before the local keep-alive timer fires. When a K-ALIVE request is received, the passive entity MUST generate a 200 OK Control Framework response and reset the local keep-alive timer. No other Control Framework response is valid. If no K-ALIVE message is received (or a transport level problem is detected by some other means) before the local keep-alive timer fires, the passive entity MUST tear down the SIP INVITE dialog and recover the associated Control Channel resources.

6.3.4. SYNC Transactions

The initial SYNC request on a Control Channel is used to negotiate the timeout period for the Control Channel keep-alive mechanism and to allow clients and servers to learn the Control Packages that each supports. Subsequent SYNC requests MAY be used to change the set of Control Packages that can be used on the Control Channel.

6.3.4.1. Timeout Negotiation for the Initial SYNC Transaction

The initial SYNC request allows the timeout period for the Control Channel keep-alive mechanism to be negotiated. The following rules MUST be followed for the initial SYNC request:

- If the Client initiating the SDP offer has a COMEDIA 'setup' attribute equal to active, the 'Keep-Alive' header MUST be included in the SYNC message generated by the offerer. The value of the 'Keep-Alive' header SHOULD be in the range of 95 to 120 seconds (this is consistent with SIP Outbound [RFC5626]). The value of the 'Keep-Alive' header MUST NOT exceed 600 seconds. The client that generated the SDP "Answer" (the passive client) MUST copy the 'Keep-Alive' header into the 200 response to the SYNC message with the same value.

- If the Client initiating the SDP offer has a COMEDIA 'setup' attribute equal to passive, the 'Keep-Alive' header parameter MUST be included in the SYNC message generated by the answerer. The value of the 'Keep-Alive' header SHOULD be in the range of 95 to

120 seconds. The client that generated the SDP offer (the passive client) MUST copy the 'Keep-Alive' header into the 200 response to the SYNC message with the same value.

- If the Client initiating the SDP offer has a COMEDIA 'setup' attribute equal to 'actpass', the 'Keep-Alive' header parameter MUST be included in the SYNC message of the entity who is the active participant in the SDP session. If the client generating the subsequent SDP answer places a value of 'active' in the COMEDIA SDP 'setup' attribute, it will generate the SYNC request and include the 'Keep-Alive' header. The value SHOULD be in the range 95 to 120 seconds. If the client generating the subsequent SDP answer places a value of 'passive' in the COMEDIA 'setup' attribute, the original UA making the SDP will generate the SYNC request and include the 'Keep-Alive' header. The value SHOULD be in the range 95 to 120 seconds.

- If the initial negotiated offer/answer results in a COMEDIA 'setup' attribute equal to 'holdconn', the initial SYNC mechanism will occur when the offer/answer exchange is updated and the active/passive roles are resolved using COMEDIA.

The previous steps ensure that the entity initiating the Control Channel connection is always the one specifying the keep-alive timeout period. It will always be the initiator of the connection who generates the K-ALIVE messages.

Once negotiated, the keep-alive timeout applies for the remainder of the Control Framework session. Any subsequent SYNC messages generated in the Control Channel do not impact the negotiated keep-alive property of the session. The 'Keep-Alive' header MUST NOT be included in subsequent SYNC messages, and if it is received, it MUST be ignored.

6.3.4.2. Package Negotiation

As part of the SYNC message exchange, a client generating the request MUST include a 'Packages' header, as defined in Section 9. The 'Packages' header contains a list of all Control Framework packages that can be supported within this control session, from the perspective of the client creating the SYNC message. All Channel Framework package names MUST be tokens that adhere to the rules set out in Section 8. The 'Packages' header of the initial SYNC message MUST contain at least one value.

A server receiving the initial SYNC request MUST examine the contents of the 'Packages' header. If the server supports at least one of the packages listed in the request, it MUST respond with a 200 response

code. The response MUST contain a 'Packages' header that lists the supported packages that are in common with those from the 'Packages' header of the request (either all or a subset). This list forms a common set of Control Packages that are supported by both parties. Any Control Packages supported by the server that are not listed in the 'Packages' header of the SYNC request MAY be placed in the 'Supported' header of the response. This provides a hint to the client that generated the SYNC request about additional packages supported by the server.

If no common packages are supported by the server receiving the SYNC message, it MUST respond with a 422 error response code. The error response MUST contain a 'Supported' header indicating the packages that are supported. The initiating client can then choose to either re-submit a new SYNC message based on the 422 response or consider the interaction a failure. This would lead to termination of the associated SIP INVITE dialog by sending a SIP BYE request, as per [RFC3261].

Once the initial SYNC transaction is completed, either client MAY choose to send a subsequent new SYNC message to re-negotiate the packages that are supported within the Control Channel. A new SYNC message whose 'Packages' header has different values from the previous SYNC message can effectively add and delete the packages used in the Control Channel. If a client receiving a subsequent SYNC message does not wish to change the set of packages, it MUST respond with a 421 Control Framework response code. Subsequent SYNC messages MUST NOT change the value of the 'Dialog-ID' and 'Keep-Alive' Control Framework headers that appeared in the original SYNC negotiation.

An entity MAY honor Control Framework commands relating to a Control Package it no longer supports after package re-negotiation. When the entity does not wish to honor such commands, it MUST respond to the request with a 420 response.

7. Response Code Descriptions

The following response codes are defined for transaction responses to methods defined in Section 6.1. All response codes in this section MUST be supported and can be used in response to both CONTROL and REPORT messages except that a 202 MUST NOT be generated in response to a REPORT message.

Note that these response codes apply to Framework Transactions only. Success or error indications for Control Commands MUST be treated as the result of a Control Command and returned in either a 200 response or REPORT message.

7.1. 200 Response Code

The framework protocol transaction completed successfully.

7.2. 202 Response Code

The framework protocol transaction completed successfully and additional information will be provided at a later time through the REPORT mechanism defined in Section 6.3.2.

7.3. 400 Response Code

The request was syntactically incorrect.

7.4. 403 Response Code

The server understood the request, but is refusing to fulfill it. The client SHOULD NOT repeat the request.

7.5. 405 Response Code

Method not allowed. The primitive is not supported.

7.6. 406 Response Code

Message out of sequence.

7.7. 420 Response Code

Intended target of the request is for a Control Package that is not valid for the current session.

7.8. 421 Response Code

Recipient does not wish to re-negotiate Control Packages at this moment in time.

7.9. 422 Response Code

Recipient does not support any Control Packages listed in the SYNC message.

7.10. 423 Response Code

Recipient has an existing transaction with the same transaction ID.

7.11. 481 Response Code

The transaction of the request does not exist. In response to a SYNC request, the 481 response code indicates that the corresponding SIP INVITE dialog usage does not exist.

7.12. 500 Response Code

The recipient does not understand the request.

8. Control Packages

Control Packages specify behavior that extends the capability defined in this document. Control Packages MUST NOT weaken statements of "MUST" and "SHOULD" strength in this document. A Control Package MAY strengthen "SHOULD", "RECOMMENDED", and "MAY" to "MUST" if justified by the specific usage of the framework.

In addition to the usual sections expected in Standards-Track RFCs and SIP extension documents, authors of Control Packages need to address each of the issues detailed in the following sub-sections. The following sections MUST be used as a template and included appropriately in all Control-Package specifications. To reiterate, the following sections do not solely form the basis of all Control-Package specifications but are included as a minimum to provide essential package-level information. A Control-Package specification can take any valid form it wishes as long as it includes at least the following information listed in this section.

8.1. Control Package Name

This section MUST be present in all extensions to this document and provides a token name for the Control Package. The section MUST include information that appears in the IANA registration of the token. Information on registering Control Package tokens is contained in Section 13.

8.2. Framework Message Usage

The Control Framework defines a number of message primitives that can be used to exchange commands and information. There are no limitations restricting the directionality of messages passed down a Control Channel. This section of a Control Package document MUST explicitly detail the types of Framework messages (Methods) that can be used as well as provide an indication of directionality between entities. This will include which role type is allowed to initiate a request type.

8.3. Common XML Support

This optional section is only included in a Control Package if the attributes for media dialog or conference reference are required, as defined and discussed in Appendix A.1. The Control Package will make strong statements (using language from RFC 2119 [RFC2119]) if the XML schema defined in Appendix A.1 is to be supported. If only part of the schema is required (for example, just 'connectionid' or 'conferenceid'), the Control Package will make equally strong statements (using language from RFC 2119 [RFC2119]).

8.4. CONTROL Message Bodies

This mandatory section of a Control Package defines the control body that can be contained within a CONTROL command request, as defined in Section 6, or that no Control Package body is required. This section MUST indicate the location of detailed syntax definitions and semantics for the appropriate MIME [RFC2045] body type that apply to a CONTROL command request and, optionally, the associated 200 response. For Control Packages that do not have a Control Package body, making such a statement satisfies the "MUST" strength of this section in the Control Package document.

8.5. REPORT Message Bodies

This mandatory section of a Control Package defines the REPORT body that can be contained within a REPORT command request, as defined in Section 6, or that no report package body is required. This section MUST indicate the location of detailed syntax definitions and semantics for the appropriate MIME [RFC2045] body type. It should be noted that the Control Framework specification does allow for payloads to exist in 200 responses to CONTROL messages (as defined in this document). An entity that is prepared to receive a payload type in a REPORT message MUST also be prepared to receive the same payload in a 200 response to a CONTROL message. For Control Packages that do not have a Control Package body, stating such satisfies the "MUST" strength of this section in the Control Package document.

8.6. Audit

Auditing of various Control Package properties such as capabilities and resources (package-level meta-information) is extremely useful. Such meta-data usually has no direct impact on Control Framework interactions but allows for contextual information to be learnt. Control Packages are encouraged to make use of Control Framework interactions to provide relevant package audit information.

This section SHOULD include the following information:

- If an auditing capability is available in this package.
- How auditing information is triggered (for example, using a Control Framework CONTROL message) and delivered (for example, in a Control Framework 200 response).
- The location of the audit query and response format for the payload (for example, it could be a separate XML schema OR part of a larger XML schema).

8.7. Examples

It is strongly RECOMMENDED that Control Packages provide a range of message flows that represent common flows using the package and this framework document.

9. Formal Syntax

9.1. Control Framework Formal Syntax

The Control Framework interactions use the UTF-8 transformation format as defined in [RFC3629]. The syntax in this section uses the Augmented Backus-Naur Form (ABNF) as defined in [RFC5234] including types 'DIGIT', 'CRLF', and 'ALPHA'.

Unless otherwise stated in the definition of a particular header field, field values, parameter names, and parameter values are not case-sensitive.

```
control-req-or-resp = control-request / control-response
control-request = control-req-start *headers CRLF [control-content]
control-response = control-resp-start *headers CRLF [control-content]
control-req-start  = pCFW SP trans-id SP method CRLF
control-resp-start = pCFW SP trans-id SP status-code CRLF

pCFW = %x43.46.57; CFW in caps
trans-id = alpha-num-token
method = mCONTROL / mREPORT / mSYNC / mK-ALIVE / other-method
mCONTROL = %x43.4F.4E.54.52.4F.4C ; CONTROL in caps
mREPORT = %x52.45.50.4F.52.54     ; REPORT in caps
mSYNC = %x53.59.4E.43             ; SYNC in caps
mK-ALIVE = %x4B.2D.41.4C.49.56.45 ; K-ALIVE in caps

other-method = 1*UPALPHA
status-code = 3*DIGIT ; any code defined in this and other documents
```

```
  headers = header-name CRLF

  header-name = (Content-Length
   /Content-Type
   /Control-Package
   /Status
   /Seq
   /Timeout
   /Dialog-ID
   /Packages
   /Supported
   /Keep-alive
   /ext-header)

  Content-Length = "Content-Length:" SP 1*DIGIT
  Control-Package = "Control-Package:" SP 1*alpha-num-token
  Status = "Status:" SP ("update" / "terminate" )
  Timeout = "Timeout:" SP 1*DIGIT
  Seq = "Seq:" SP 1*DIGIT
  Dialog-ID = "Dialog-ID:" SP dialog-id-string
  Packages = "Packages:" SP package-name *(COMMA package-name)
  Supported = "Supported:" SP supprtd-alphanum *(COMMA supprtd-alphanum)
  Keep-alive = "Keep-Alive:" SP kalive-seconds

  dialog-id-string = alpha-num-token
  package-name = alpha-num-token
  supprtd-alphanum = alpha-num-token
  kalive-seconds = 1*DIGIT

  alpha-num-token = ALPHANUM  3*31alpha-num-tokent-char
  alpha-num-tokent-char = ALPHANUM / "." / "-" / "+" / "%" / "=" / "/"

  control-content = *OCTET

  Content-Type = "Content-Type:" SP media-type
  media-type = type "/" subtype *(SP ";" gen-param )
  type = token    ; Section 4.2 of RFC 4288
  subtype = token ; Section 4.2 of RFC 4288

  gen-param = pname [ "=" pval ]
  pname = token
  pval  = token / quoted-string

  token = 1*(%x21 / %x23-27 / %x2A-2B / %x2D-2E
             / %x30-39 / %x41-5A / %x5E-7E)
```

```
quoted-string = DQUOTE *(qdtext / qd-esc) DQUOTE
qdtext = SP / HTAB / %x21 / %x23-5B / %x5D-7E
           / UTF8-NONASCII
qd-esc = (BACKSLASH BACKSLASH) / (BACKSLASH DQUOTE)
BACKSLASH = "\"
UPALPHA  = %x41-5A
ALPHANUM = ALPHA / DIGIT

ext-header = hname ":" SP hval CRLF

hname = ALPHA *token
hval = utf8text

utf8text = *(HTAB / %x20-7E / UTF8-NONASCII)

UTF8-NONASCII = UTF8-2 / UTF8-3 / UTF8-4 ; From RFC 3629
```

The following table details a summary of the headers that can be contained in Control Framework interactions.

Header field	Where	CONTROL	REPORT	SYNC	K-ALIVE
Content-Length		o	o	-	-
Control-Package	R	m	-	-	-
Seq		-	m	-	-
Status	R	-	m	-	-
Timeout	R	-	m	-	-
Timeout	202	-	m	-	-
Dialog-ID	R	-	-	m	-
Packages		-	-	m	-
Supported	r	-	-	o	-
Keep-Alive	R	-	-	o	-
Content-Type		o	o	-	-

Table 1: Summary of Headers in Control Framework Interactions

The notation used in Table 1 is as follows:

R: header field may only appear in requests.
r: header field may only appear in responses.
2xx, 4xx, etc.: response codes with which the header field can be used.
[blank]: header field may appear in either requests or responses.
m: header field is mandatory.
o: header field is optional.
-: header field is not applicable (ignored if present).

9.2. Control Framework Dialog Identifier SDP Attribute

This specification defines a new media-level value attribute: 'cfw-id'. Its formatting in SDP is described by the following ABNF [RFC5234].

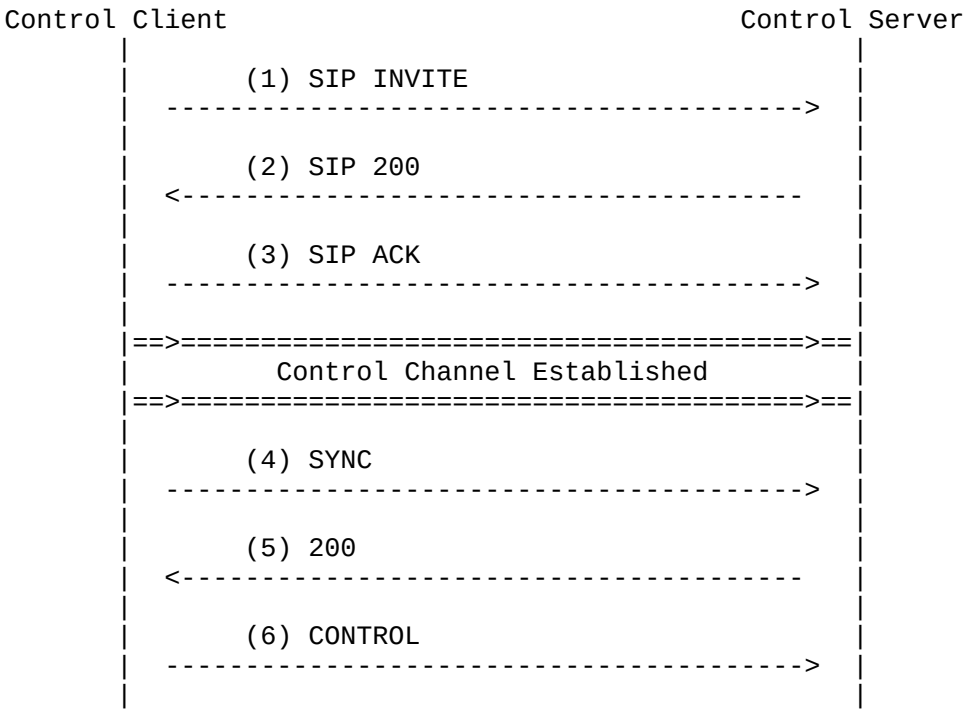
```
cfw-dialog-id = "a=cfw-id:" 1*(SP cfw-id-name) CRLF

cfw-id-name   = token

token         = 1*(token-char)

token-char    = %x21 / %x23-27 / %x2A-2B / %x2D-2E / %x30-39
                / %x41-5A / %x5E-7E
```

The token-char and token elements are defined in [RFC4566] but included here to provide support for the implementer of this SDP feature.

10. Examples

The following examples provide an abstracted flow of Control Channel establishment and Control Framework message exchange. The SIP signaling is prefixed with the token 'SIP'. All other messages are Control Framework interactions defined in this document.

In this example, the Control Client establishes a Control Channel, SYNCs with the Control Server, and issues a CONTROL request that can't be completed within the 'Transaction-Timeout', so the Control Server returns a 202 response code to extend the transaction. The Control Server then follows with REPORTs until the requested action has been completed. The SIP INVITE dialog is then terminated.



(1) Control Client-->Control Server (SIP): INVITE
sip:control-server@example.com

```
INVITE sip:control-server@example.com SIP/2.0
To: <sip:control-server@example.com>
From: <sip:control-client@example.com>;tag=8937498
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK123
CSeq: 1 INVITE
Max-Forwards: 70
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-client@pc1.example.com>
Content-Type: application/sdp
Content-Length: 206

v=0
o=originator 2890844526 2890842808 IN IP4 controller.example.com
s=-
c=IN IP4 control-client.example.com
m=application 49153 TCP cfw
a=setup:active
a=connection:new
a=cfw-id:fndskuhHKsd783hjdla
```

(2) Control Server-->Control Client (SIP): 200 OK

```
SIP/2.0 200 OK
To: <sip:control-server@example.com>;tag=023983774
From: <sip:control-client@example.com>;tag=8937498
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK123;received=192.0.2.5
CSeq: 1 INVITE
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-server@pc2.example.com>
Content-Type: application/sdp
Content-Length: 203

v=0
o=responder 2890844600 2890842900 IN IP4 controller.example.com
s=-
c=IN IP4 control-server.example.com
m=application 49153 TCP cfw
a=setup:passive
a=connection:new
a=cfw-id:7JeDi23i7eiysi32
```

(3) Control Client-->Control Server (SIP): ACK

(4) Control Client opens a TCP connection to the Control Server. The connection can now be used to exchange Control Framework messages. Control Client-->Control Server (Control Framework message): SYNC.

```
CFW 8djae7khauj SYNC
Dialog-ID: fndskuhHKsd783hjdla
Keep-Alive: 100
Packages: msc-ivr-basic/1.0
```

(5) Control Server-->Control Client (Control Framework message): 200.

```
CFW 8djae7khauj 200
Keep-Alive: 100
Packages: msc-ivr-basic/1.0
Supported: msc-ivr-vxml/1.0,msc-conf-audio/1.0
```

(6) Once the SYNC process has completed, the connection can now be used to exchange Control Framework messages. Control Client-->Control Server (Control Framework message): CONTROL.

```
CFW i387yeiqyiq CONTROL
Control-Package: <package-name>
Content-Type: example_content/example_content
```

```
   Content-Length: 11

   <XML BLOB/>
```

(7) Control Server-->Control Client (Control Framework message): 202.

```
   CFW i387yeiqyiq 202
   Timeout: 10
```

(8) Control Server-->Control Client (Control Framework message): REPORT.

```
   CFW i387yeiqyiq REPORT
   Seq: 1
   Status: update
   Timeout: 10
```

(9) Control Client-->Control Server (Control Framework message): 200.

```
   CFW i387yeiqyiq 200
   Seq: 1
```

(10) Control Server-->Control Client (Control Framework message): REPORT.

```
   CFW i387yeiqyiq REPORT
   Seq: 2
   Status: update
   Timeout: 10
   Content-Type: example_content/example_content
   Content-Length: 11

   <XML BLOB/>
```

(11) Control Client-->Control Server (Control Framework message): 200.

```
   CFW i387yeiqyiq 200
   Seq: 2
```

(12) Control Server-->Control Client (Control Framework message): REPORT.

```
   CFW i387yeiqyiq REPORT
   Seq: 3
   Status: terminate
```

```
   Timeout: 10
   Content-Type: example_content/example_content
   Content-Length: 11

   <XML BLOB/>
```

(13) Control Client-->Control Server (Control Framework message): 200.

```
   CFW i387yeiqyiq 200
   Seq: 3
```

(14) Control Client-->Control Server (SIP): BYE

```
   BYE sip:control-server@pc2.example.com SIP/2.0
   To: <sip:control-server@example.com>;tag=023983774
   From: <sip:client@example.com>;tag=8937498
   Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK234
   CSeq: 2 BYE
   Max-Forwards: 70
   Call-ID: 893jhoeihjr8392@example.com
   Contact: <sip:control-client@pc1.example.com>
   Content-Length: 0
```

(15) Control Server-->Control Client (SIP): 200 OK

```
SIP/2.0 200 OK
To: <sip:control-server@example.com>;tag=023983774
From: <sip:client@example.com>;tag=8937498
Via: SIP/2.0/UDP client.example.com;branch=z9hG4bK234;received=192.0.2.5
CSeq: 2 BYE
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-server@pc1.example.com>
Content-Length: 0
```

11. Extensibility

The Media Control Channel Framework was designed to be only minimally extensible. New methods, header fields, and status codes can be defined in Standards-Track RFCs. The Media Control Channel Framework does not contain a version number or any negotiation mechanism to require or discover new features. If an extension is specified in the future that requires negotiation, the specification will need to describe how the extension is to be negotiated in the encapsulating signaling protocol. If a non-interoperable update or extension occurs in the future, it will be treated as a new protocol, and it MUST describe how its use will be signaled.

In order to allow extension header fields without breaking interoperability, if a Media Control Channel device receives a request or response containing a header field that it does not understand, it MUST ignore the header field and process the request or response as if the header field was not present. If a Media Control Channel device receives a request with an unknown method, it MUST return a 500 response.

12. Security Considerations

The Channel Framework provides confidentiality and integrity for the messages it transfers. It also provides assurances that the connected host is the host that it meant to connect to and that the connection has not been hijacked, as discussed in the remainder of this section.

In design, the Channel Framework complies with the security-related requirements documented in "Media Server Control Protocol Requirements" [RFC5167] -- more specifically, REQ-MCP-11, REQ-MCP-12, REQ-MCP-13, and REQ-MCP-14. Specific security measures employed by the Channel Framework are summarized in the following sub-sections.

12.1. Session Establishment

Channel Framework sessions are established as media sessions described by SDP within the context of a SIP INVITE dialog. In order to ensure secure rendezvous between Control Framework clients and servers, the Media Channel Control Framework should make full use of mechanisms provided by SIP. The use of the 'cfw-id' SDP attribute results in important session information being carried across the SIP network. For this reason, SIP clients using this specification MUST use appropriate security mechanisms, such as TLS [RFC5246] and SMIME [RFC5751], when deployed in open networks.

12.2. Transport-Level Protection

When using only TCP connections, the Channel Framework security is weak. Although the Channel Framework requires the ability to protect this exchange, there is no guarantee that the protection will be used all the time. If such protection is not used, anyone can see data exchanges.

Sensitive data, such as private and financial data, is carried over the Control Framework channel. Clients and servers must be properly authenticated/authorized and the Control Channel must permit the use of confidentiality, replay protection, and integrity protection for the data. To ensure Control Channel protection, Control Framework clients and servers MUST support TLS and SHOULD use it by default

unless alternative Control Channel protection is used or a protected environment is guaranteed by the administrator of the network. Alternative Control Channel protection MAY be used if desired (e.g., IPsec [RFC5246]).

TLS is used to authenticate devices and to provide integrity, replay protection, and confidentiality for the header fields being transported on the Control Channel. Channel Framework elements MUST implement TLS and MUST also implement the TLS ClientExtendedHello extended hello information for server name indication as described in [RFC5246]. A TLS cipher-suite of TLS_RSA_WITH_AES_128_CBC_SHA [RFC3261] MUST be supported. Other cipher-suites MAY also be supported.

When a TLS client establishes a connection with a server, it is presented with the server's X.509 certificate. Authentication proceeds as described in Section 7.3 ("Client Behavior") of RFC 5922 [RFC5922].

A TLS server conformant to this specification MUST ask for a client certificate; if the client possesses a certificate, it will be presented to the server for mutual authentication, and authentication proceeds as described in Section 7.4 ("Server Behavior") of RFC 5922 [RFC5922].

12.3. Control Channel Policy Management

This specification permits the establishment of a dedicated Control Channel using SIP. It is also permitted for entities to create multiple channels for the purpose of failover and redundancy. As a general solution, the ability for multiple entities to create connections and have access to resources could be the cause of potential conflict in shared environments. It should be noted that this document does not carry any specific mechanism to overcome such conflicts but will provide a summary of how to do so.

It can be determined that access to resources and use of Control Channels relate to policy. It can be considered implementation and deployment detail that dictates the level of policy that is adopted. The authorization and associated policy of a Control Channel can be linked to the authentication mechanisms described in this section. For example, strictly authenticating a Control Channel using TLS authentication allows entities to protect resources and ensure the required level of granularity. Such policy can be applied at the package level or even as low as a structure like a conference instance (Control Channel X is not permitted to issue commands for Control Package y OR Control Channel A is not permitted to issue commands for conference instance B). Systems should ensure that, if

required, an appropriate policy framework is adopted to satisfy the requirements for implemented packages. The most robust form of policy can be achieved using a strong authentication mechanism such as mutual TLS authentication on the Control Channel. This specification provides a Control Channel response code (403) to indicate to the issuer of a command that it is not permitted. The 403 response MUST be issued to Control Framework requests that are not permitted under the implemented policy. If a 403 response is received, a Control Framework client MAY choose to re-submit the request with differing requirements or to abandon the request. The 403 response does not provide any additional information on the policy failure due to the generic nature of this specification. Individual Control Packages can supply additional information if required. The mechanism for providing such additional information is not mandated in this specification. It should be noted that additional policy requirements to those covered in this section might be defined and applied in individual packages that specify a finer granularity for access to resources, etc.

13. IANA Considerations

IANA has created a new registry for SIP Control Framework parameters. The "Media Control Channel Framework Parameters" registry is a container for sub-registries. This section further introduces sub-registries for control packages, method names, status codes, header field names, and port and transport protocol.

Additionally, Section 13.6 registers a new MIME type for use with SDP.

For all registries and sub-registries created by this document, the policy applied when creating a new registration is also applied when changing an existing registration.

13.1. Control Packages Registration Information

This specification establishes the Control Packages sub-registry under Media Control Channel Framework Packages. New parameters in this sub-registry must be published in an RFC (either in the IETF stream or Independent Submission stream), using the IANA policy [RFC5226] "RFC Required".

As this document specifies no package or template-package names, the initial IANA registration for Control Packages will be empty. The remainder of the text in this section gives an example of the type of information to be maintained by the IANA.

The table below lists the Control Packages defined in the "Media Control Channel Framework".

Package Name	Reference
example1	[RFCXXXX]

13.1.1. Control Package Registration Template

Package Name:

(Package names must conform to the syntax described in Section 8.1.)

Published Specification(s):

(Control Packages require an RFC.)

Person & email address to contact for further information:

13.2. Control Framework Method Names

This specification establishes the Method Names sub-registry under Media Control Channel Framework Parameters and initiates its population as follows. New parameters in this sub-registry must be published in an RFC (either in the IETF stream or Independent Submission stream).

CONTROL - [RFC6230]
REPORT - [RFC6230]
SYNC - [RFC6230]
K-ALIVE - [RFC6230]

The following information MUST be provided in an RFC in order to register a new Control Framework method:

- The method name.
- The RFC number in which the method is registered.

13.3. Control Framework Status Codes

This specification establishes the Status Code sub-registry under Media Control Channel Framework Parameters. New parameters in this sub-registry must be published in an RFC (either in the IETF stream or Independent Submission stream). Its initial population is defined in Section 9. It takes the following format:

Code Description Reference

The following information MUST be provided in an RFC in order to register a new Control Framework status code:

- The status code number.
- The RFC number in which the method is registered.
- A brief description of the status code.

13.4. Control Framework Header Fields

This specification establishes the Header Field sub-registry under Media Control Channel Framework Parameters. New parameters in this sub-registry must be published in an RFC (either in the IETF stream or Independent Submission stream). Its initial population is defined as follows:

Control-Package - [RFC6230]
Status - [RFC6230]
Seq - [RFC6230]
Timeout - [RFC6230]
Dialog-ID - [RFC6230]
Packages - [RFC6230]
Supported - [RFC6230]
Keep-Alive - [RFC6230]
Content-Type - [RFC6230]
Content-Length - [RFC6230]

The following information MUST be provided in an RFC in order to register a new Channel Framework header field:

- The header field name.
- The RFC number in which the method is registered.

13.5. Control Framework Port

The Control Framework uses TCP port 7563, from the "registered" port range. Usage of this value is described in Section 4.1.

13.6. Media Type Registrations

This section describes the media types and names associated with payload formats used by the Control Framework. The registration uses the templates defined in [RFC4288]. It follows [RFC4855].

13.6.1. Registration of MIME Media Type application/cfw

Type name: application

Subtype name: cfw

Required parameters: None

Optional parameters: None

Encoding considerations: Binary and see Section 4 of RFC 6230

Security considerations: See Section 12 of RFC 6230

Interoperability considerations:
Endpoints compliant to this specification must use this MIME type. Receivers who cannot support this specification will reject using appropriate protocol mechanism.

Published specification: RFC 6230

Applications that use this media type:
Applications compliant with Media Control Channels.

Additional Information:
Magic number(s): (none)
File extension(s): (none)
Macintosh file type code(s): (none)

Person & email address to contact for further information:
Chris Boulton <chris@ns-technologies.com>

Intended usage: COMMON

Restrictions on usage:
Should be used only in conjunction with this specification, RFC 6230.

Author: Chris Boulton

Change controller:
IETF MEDIACTRL working group, delegated from the IESG.

13.6.2. Registration of MIME Media Type application/framework-attributes+xml

Type name: application

Subtype name: framework-attributes+xml

Required parameters: (none)

Optional parameters: Same as charset parameter of application/xml as specified in RFC 3023 [RFC3023].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [RFC3023].

Security considerations: No known security considerations outside of those provided by core Media Control Channel Framework.

Interoperability considerations: This content type provides common constructs for related Media Control Channel packages.

Published specification: RFC 6230

Applications that use this media type: Implementations of appropriate Media Control Channel packages.

Additional information:
Magic number(s): (none)
File extension(s): (none)
Macintosh file type code(s): (none)

Person & email address to contact for further information:
Chris Boulton <chris@ns-technologies.com>

Intended usage: LIMITED USE

Author/Change controller: The IETF

Other information: None.

13.7. 'cfw-id' SDP Attribute

Contact name:	Chris Boulton <chris@ns-technologies.com>
Attribute name:	"cfw-id".
Type of attribute	Media level.

Subject to charset: Not.

Purpose of attribute: The 'cfw-id' attribute indicates an identifier that can be used to correlate the Control Channel with the SIP INVITE dialog used to negotiate it, when the attribute value is used within the Control Channel.

Allowed attribute values: A token.

13.8. URN Sub-Namespace for urn:ietf:params:xml:ns:control:framework-attributes

IANA has registered a new XML namespace, "urn:ietf:params:xml:ns:control:framework-attributes", per the guidelines in RFC 3688 [RFC3688].

URI: urn:ietf:params:xml:ns:control:framework-attributes

Registrant Contact: IETF MEDIACTRL working group <mediactrl@ietf.org>, Chris Boulton <chris@ns-technologies.com>.

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
   <title>Media Control Channel attributes</title>
  </head>
  <body>
   <h1>Namespace for Media Control Channel attributes</h1>
   <h2>urn:ietf:params:xml:ns:control:framework-attributes</h2>
     <p>See <a href="http://www.rfc-editor.org/rfc/rfc6230.txt">
        RFC 6230</a>.</p>
  </body>
 </html>
END
```

13.9. XML Schema Registration

This section registers an XML schema as per the guidelines in RFC 3688 [RFC3688].

URI: urn:ietf:params:xml:ns:control:framework-attributes

Registrant Contact: IETF MEDIACTRL working group <mediactrl@ietf.org>, Chris Boulton <chris@ns-technologies.com>.

Schema: The XML for this schema can be found in Appendix A.1 of this document.

14. Contributors

Asher Shiratzky from Radvision provided valuable support and contributions to the early versions of this document.

15. Acknowledgments

The authors would like to thank Ian Evans of Avaya, Michael Bardzinski and John Dally of NS-Technologies, Adnan Saleem of Radisys, and Dave Morgan for useful review and input to this work. Eric Burger contributed to the early phases of this work.

Expert review was also provided by Spencer Dawkins, Krishna Prasad Kalluri, Lorenzo Miniero, and Roni Even. Hadriel Kaplan provided expert guidance on the dialog association mechanism. Lorenzo Miniero has constantly provided excellent feedback based on his work.

Ben Campbell carried out the RAI expert review on this document and provided a great deal of invaluable input. Brian Weis carried out a thorough security review. Jonathan Lennox carried out a thorough SDP review that provided some excellent modifications. Text from Eric Burger was used in the introduction in the explanation for using SIP.

16. References

16.1. Normative References

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3262] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, October 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, February 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5922] Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain Certificates in the Session Initiation Protocol (SIP)", RFC 5922, June 2010.

16.2. Informative References

[MSCL-THOUGHTS] Burger, E., "Media Server Control Language and Protocol Thoughts", Work in Progress, June 2006.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3725] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, April 2004.

[RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[RFC3841] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", RFC 3841, August 2004.

[RFC5125] Taylor, T., "Reclassification of RFC 3525 to Historic", RFC 5125, February 2008.

[RFC5167] Dolly, M. and R. Even, "Media Server Control Protocol Requirements", RFC 5167, March 2008.

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

Appendix A. Common Package Components

During the creation of the Control Framework, it has become clear that there are a number of components that are common across multiple packages. It has become apparent that it would be useful to collect such reusable components in a central location. In the short term, this appendix provides the placeholder for the utilities, and it is the intention that this section will eventually form the basis of an initial 'Utilities Document' that can be used by Control Packages.

A.1. Common Dialog/Multiparty Reference Schema

The following schema provides some common attributes for allowing Control Packages to apply specific commands to a particular SIP media dialog (also referred to as "Connection") or conference. If used within a Control Package, the Connection and multiparty attributes will be imported and used appropriately to specifically identify either a SIP dialog or a conference instance. If used within a package, the value contained in the 'connectionid' attribute MUST be constructed by concatenating the 'Local' and 'Remote' SIP dialog identifier tags as defined in [RFC3261]. They MUST then be separated using the ':' character. So the format would be:

'Local Dialog tag' + ':' + 'Remote Dialog tag'

As an example, for an entity that has a SIP Local dialog identifier of '7HDY839' and a Remote dialog identifier of 'HJKSkyHS', the 'connectionid' attribute for a Control Framework command would be:

7HDY839:HJKSkyHS

It should be noted that Control Framework requests initiated in conjunction with a SIP dialog will produce a different 'connectionid' value depending on the directionality of the request; for example, Local and Remote tags are locally identifiable.

As with the Connection attribute previously defined, it is useful to have the ability to apply specific Control Framework commands to a number of related dialogs, such as a multiparty call. This typically consists of a number of media dialogs that are logically bound by a single identifier. The following schema allows for Control Framework commands to explicitly reference such a grouping through a 'conferenceid' XML container. If used by a Control Package, any control XML referenced by the attribute applies to all related media dialogs. Unlike the dialog attribute, the 'conferenceid' attribute does not need to be constructed based on the overlying SIP dialog. The 'conferenceid' attribute value is system specific and should be selected with relevant context and uniqueness.

It should be noted that the values contained in both the 'connectionid' and 'conferenceid' identifiers MUST be compared in a case-sensitive manner.

The full schema follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
  targetNamespace="urn:ietf:params:xml:ns:control:framework-attributes"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:ietf:params:xml:ns::control:framework-attributes"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

       <xsd:attributeGroup name="framework-attributes">
         <xsd:annotation>
           <xsd:documentation>
             SIP Connection and Conf Identifiers
           </xsd:documentation>
         </xsd:annotation>

         <xsd:attribute name="connectionid" type="xsd:string"/>

         <xsd:attribute name="conferenceid" type="xsd:string"/>

       </xsd:attributeGroup>
</xsd:schema>
```

Authors' Addresses

Chris Boulton
NS-Technologies

EMail: chris@ns-technologies.com

Tim Melanchuk
Rainwillow

EMail: timm@rainwillow.com

Scott McGlashan
Hewlett-Packard
Gustav III:s boulevard 36
SE-16985 Stockholm, Sweden

EMail: smcg.stds01@mcglashan.org