Network Working Group C. Boulton
Internet-Draft Avaya
Expires: August 25, 2008 T. Melanchuk
Rain Willow Communications
S. McGlashan
Hewlett-Packard
February 22, 2008

Media Control Channel Framework
draft-ietf-mediactrl-sip-control-framework-01

Status of this Memo

Copyright Notice

Abstract

This document describes a Framework and protocol for application deployment where the application logic and processing are distributed. 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 distributed, centralized conference system, as defined by the IETF. It is not, however, limited to this scope and it is envisioned that this generic Framework will be used for a wide variety of de-coupled control architectures between network entities.

Table of Contents

1. Introduction

Real-time media applications are often developed using an architecture where the application logic and processing activities are distributed. Commonly, the application logic runs on "application servers" whilst 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 the 'Media Server Control Protocol Requirements' document[8]. While the Framework is not media server control specific, it is the primary driver and use case for this work. It is intended that the framework contained in this document will be used for a plethora of appropriate device control scenarios.

This document does not define a SIP based extension that can be used directly for the control of external components. The framework mechanism must be extended by other documents that are known as "Control Packages". A comprehensive set of guidelines for creating "Control Packages" is described in Section 8.

Current IETF device control protocols, such as megaco [7], while excellent for controlling media gateways that bridge separate networks, are troublesome for supporting media-rich applications in SIP networks, because they duplicate many of the functions inherent in SIP. Rather than relying on single protocol session establishment, application developers need to translate between two separate mechanisms.

Application servers traditionally use SIP third party call control RFC 3725 [12] to establish media sessions from SIP user agents to a media server. SIP, as defined in RFC 3261 [2], also 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

In this document, BCP 14/RFC 2119 [1] defines the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL". In addition, BCP 15 indicates requirement levels for compliant implementations.

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

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 in this case, has a direct RTP [15] 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 may 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: SYNCH, 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 MUST complete within 'Transaction-Timeout' time.

extended transaction lifetime: An extended transaction lifetime is used to extend the lifetime of a CONTROL method transaction when the Control Command it carries cannot be completed within Transaction-Timeout milliseconds. A Server extends the lifetime of a CONTROL method transaction by sending a 202 response code followed by one or more REPORT transactions as specified in Section 6.1.2. Extended transaction lifetimes allow command failures to be discovered at the transaction layer.

Transaction-Timeout: the maximum allowed time between a control Client or Server issuing a framework message and receiving a corresponding response. The value for the timeout should be based on a multiple of the network RTT plus 'Transaction-Timeout' milliseconds to allow for message parsing and processing.

[Editors Note:DP0 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

3. Overview

This document details mechanisms for establishing, using, and terminating a reliable channel using SIP for the purpose of 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 proposed 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.

The use of SIP for the specified mechanism provides many inherent capabilities which include:-

- o Service location - Use SIP Proxies or Back-to-Back User Agents for discovering Control Servers.
- o Security mechanisms - Leverage established security mechanisms such as Transport Layer Security (TLS) and Client Authentication.
- o Connection maintenance - The ability to re-negotiate a connection, ensure it is active, audit parameters, and so forth.
- o 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 at both a routing level, where RFC 3263 [4] provides the physical location of devices, and at the Service level, using Caller Preferences[13] and Callee Capabilities[14]. 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 of this document.

```
          +--------------SIP Traffic--------------+
          |                                       |
          v                                       v
       +-----+                                 +--+--+
       | SIP |                                 | SIP |
       |Stack|                                 |Stack|
   +---+-----+---+                         +---+-----+---+
   |   Control   |                         |   Control   |
   |   Client    |<----Control Channel---->|   Server    |
   +-------------+                         +-------------+
```

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 multiple control channels using separate SIP dialogs to be established 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 agnostic to any media sessions. Specific media session information can be incorporated in control interaction commands (which themselves are defined in external packages) using the XML schema defined in Section 16. 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 RFC 3261 [2], to the external Server. The SDP payload includes the required information for control channel negotiation and is the primary mechanism for conveying support for this specification (through the media type). The COMEDIA [6] specification for setting up and maintaining reliable connections is used as part of the negotiation mechanism (more detail available in later sections).

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=z9hG4bK72dhjsU
Call-ID: 7823987HJHG6
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 7575 TCP/SCFW
a=setup:active
a=connection:new
```

On receiving the INVITE request, the external Server supporting this mechanism generates a 200 OK response containing appropriate SDP.

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=z9hG4bK72dhjsU
Call-ID: 7823987HJHG6
CSeq: 1 INVITE
Contact: <sip:External-Server@servermachine.example.com>
Content-Type: application/sdp
Content-Length: [..]

v=0
o=originator 2890844526 2890842808 IN IP4 server.example.com
s=-
c=IN IP4 mserver.example.com
m=application 7563 TCP/SCFW
a=setup:passive
a=connection:new
```

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 part in the new connection.

Once established, the newly created connection can be used to exchange control language request and response primitives. If required, after the control channel has been setup, media sessions can be established using standard SIP third party call control.

Figure 4 provides a simplified example where the proposed framework is used to control a User Agent's RTP session. (1) in brackets represents the SIP dialog and dedicated control channel previously described in this overview section.

```
                      +--------Control SIP Dialog(1)---------+
                      |                                      |
                      v                                      v
                   +-----+                                +--+--+
 +------(2)------>| SIP |---------------(2)------------->| SIP |
 |                 |Stack|                                |Stack|
 |             +---+-----+---+                        +---+-----+---+
 |             |             |                        |             |
 |             |   Control   |<--Control Channel(1)-->|             |
 |             |   Client    |                        |   Control   |
 |             +-------------+                        |   Server    |
+--+--+                                               |             |
|User |                                               |             |
|Agent|<=====================RTP(2)===================>|             |
+-----+                                               +-------------+
```

Figure 4: Participant Architecture

(2) from Figure 4 represents the User Agent SIP dialog interactions and associated media flow. A User Agent would create a SIP dialog with the Control Client entity. The Control Client entity will also create a related dialog to the Control Server (B2BUA type functionality). Using the interaction illustrated by (2), the User Agent is able to negotiate media capabilities with the Control Server using standard SIP mechanisms as defined in RFC 3261 [2] and RFC 3264 [5].

4. Control Client SIP UAC Behavior - Control Channel Setup

On creating a new SIP INVITE request for control channel setup, a UAC MUST construct the protocol message as defined in RFC 3261 [2].

If a reliable response is received (as defined RFC 3261 [2] and RFC 3262 [3]), the mechanisms defined in this document are applicable to the newly created dialog.

The UAC MAY include a valid session description (an 'offer' as defined in RFC 3264 [5]) in an INVITE request using the Session Description Protocol defined in [9]. The following information defines the composition of some specific elements of the SDP payload that MUST be adhered to for compliancy to this specification when used in an SIP SDP offer.

The Connection Data line in the SDP payload is constructed as specified in [9]:

```
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 originators address, for example dns/IP representation. The address will be the network address used for connections in this specification.

Example:

```
c=IN IP4 controller.example.com
```

The SDP MUST contain a corresponding Media Description entry for compliance to this specification:

```
m=<media> <port> <proto>
```

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 constructing client can't receive incoming connections it MUST still enter a valid port range entry. The use of the port value '0' has the same meaning as defined in the SDP specification[9]. The third sub-field, <proto>, MUST equal a transport value defined in Section 12.6. All implementations compliant to this specification MUST support the value "TCP/SCFW", "TCP/TLS/SCFW", "SCTP/SCFW" and "SCTP/TLS/SCFW" as defined in Section 12.6 of this document. Implementations MUST support TLS as a transport-level security mechanism, although use of TLS in specific deployments is optional. MEDIACTRL implementations MUST support TCP as a transport protocol. MEDIACTRL implementations MAY support SCTP

as a transport protocol. When an entity identifies one of the transport values defined in Section 12.6 but is not willing to establish the session, it MUST respond using the appropriate SIP mechanism.

The SDP MUST also contain a number of SDP media attributes(a=) that are specifically defined in the COMEDIA [6] specification. The attributes provide connection negotiation and maintenance parameters. A client conforming to this specification SHOULD support all the possible values defined for media attributes from the COMEDIA [6] specification but MAY choose not to support values if it can definitely determine they will never be used (for example will only ever initiate outgoing connections). 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 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 RFC 3261 [2]), it can be sent to the appropriate location. The SIP dialog and appropriate control connection is then established.

As mentioned previously, the SIP Control Framework can be used in conjunction with other media dialogs (for example, use the control channel to play a prompt to media dialog X). For SIP based media dialogs, if not present in the SDP received by the Control Client (when acting as a B2BUA) from the User Agent, a media label SDP attribute, which is defined in RFC 4574 [10], should be inserted for every media description (identified as m= line as defined in [9]) before forwarding. This provides flexibility for the Control Client as it can generate control messages using the Control Channel that specify a particular Media stream (between User Agent and Control Server) within a SIP media dialog. If a Media label is not included in the control message, commands apply to all media associated with the dialog.

A non-2xx class error (4xx, 5xx and 6xx) SIP response received for the INVITE request indicates that no SIP dialog has been created and

is treated as specified RFC 3261 [2]. 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 like "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 should indicate to the client generating the offer that this could be due to the receiving client not supporting this specification. The client generating the offer should act as it would normally on receiving this response, as per RFC 3261 [2]. Media streams can also be rejected by setting the port to "0" in the "m=" line of the session description. A client using this specification should be prepared to receive an answer where the "m=" line it inserted for using the Control Framework has been set to "0".

4.1. Control Client SIP UAC Behavior - Media Dialogs

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 specific Control package commands to co-existing SIP dialogs that have been established with the same remote server, for example the manipulation of audio dialogs connected to a media server.

Such co-existing dialogs will pass through the Control Client (see Figure 4) entity and may contain more than one Media Description (as defined by "m=" in the SDP). The Control Client SHOULD include a media label attribute (B2BUA functionality), as defined in [10], for each "m=" definition. A Control Client constructing the SDP MAY choose not to include the media label SDP attribute if it does not require direct control on a per media stream basis.

This framework identifies the common re-use of referencing media dialogs and has specified a connection reference attribute that can optionally be imported into any Control Package. It is intended that this will reduce repetitive specifying of dialog reference language. The schema can be found in Section 16.1 in Appendix A.

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 re-used (for example playing a prompt to all participants in a Conference). The schema for such operations can also be found in Section 16.1 in Appendix A.

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

5. Control Server SIP UAS Behavior - Control Channel Setup

On receiving a SIP INVITE request, an external Server(UAS) inspects the message for indications of support for the mechanisms defined in this specification. This is achieved through inspection of the Sessions Description of the SIP INVITE message and identifying support for the appropriate media type. If the external Server wishes to construct a reliable response that conveys support for the extension, it should follow the mechanisms defined in RFC 3261 [2]. If support is conveyed in a reliable SIP provisional response, the mechanisms in RFC 3262 [3] 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 RFC 3261 [2], RFC 3262 [3], and RFC 3264 [5]

When constructing an answer, the SDP payload MUST be constructed using the semantics(Connection, Media and attribute) defined in Section 4 using valid local settings and also with full compliance to the COMEDIA[6] specification. For example, the SDP attributes included in the answer constructed for the example offer provided in Section 4 would look as illustrated below:

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

Once the SIP success response 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 Dialog 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 (SIP BYE request). The Control Channel therefore lives for the duration of the SIP dialog.

If the UAS does not support the extension defined in this document, as identified by the media contained in the Session Description, it SHOULD respond as detailed in RFC 3261 [2] with a "SIP 488" response code. If multiple media descriptions exist it MAY choose to continue processing the request and mark the port field equal to "0".

A SIP entity receiving a SIP OPTIONS request MUST respond appropriately as defined in RFC 3261 [2]. This involves providing information relating to supported SIP extensions and media types in a

200 OK response. For this extension the media types supported MUST be included in the SIP 200 OK response in a SIP "Accept" header to indicate a valid media type.

6. Control Framework Interactions

The use of the COMEDIA specification in this document allows for a Control Channel to be set up in either direction as a result of the SIP INVITE transaction. While providing a flexible negotiation mechanism, it does provide certain correlation problems between the channel and the overlying SIP dialog. Remember that the two are implicitly linked and so need a robust correlation mechanism. 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 dialog as it could be simply listening for all incoming connections on a specific port. As a consequence, some rules are applied to allow a connecting (defined as 'active' role in COMEDIA) active UA to identify the associated SIP dialog that triggered the connection. The following steps provide an identification mechanism that MUST be carried out before any other signaling is carried out on the newly created Control channel.

- o Once the connection has been established, the active UA initiating the connection (as determined by COMEDIA) MUST immediately send a Control Framework SYNCH request. The SYNCH request will be constructed as defined in Section 9.1 and MUST contain the message header, 'Dialog-ID', which contains the SIP dialog information.
- o The 'Dialog-ID' message header is constructed by concatenating the Local-tag, Call-ID and Remote-tag (as defined in Section 9.1) from the SIP dialog and separating with a '~'. See syntax defined in Section 16.1 in Appendix A and examples in Section 8.7. For example, if the SIP dialog had values of 'Local-tag=HKJDH', 'Remote-tag=JJSUSHJ' and 'Call-ID=8shKUHSUKHW@example.com' - the 'Dialog-ID' header would look like this: 'Dialog-ID=HKJDH~8shKUHSUKHW@example.com~JJSUSHJ'.
- o On creating the SYNCH request the controlling active UA MUST follow the procedures outlined in Section 6.1.3 . This provides details of connection keep-alive messages.
- o On creating the SYNCH request the controlling active UA MUST also follow the procedures outlined in Section 6.1.4. 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.
- o The active UA who initiated the connection MUST then send the SYNCH request. It MUST then wait for a period of at least 5 seconds to receive a response. It MAY choose a longer time to wait but it should not be shorter than 5 seconds.

- If no response is received for the SYNCH control message, a timeout occurs and the control channel is terminated along with the associated SIP dialog (issue a BYE request).
- If the active UA who initiated a connection receives a 481 response, this implies that the SYNCH request was received but no associated SIP dialog exists. This also results in the control channel being terminated along with the associated SIP dialog (issue a BYE request).
- All other error responses received for the SYNCH request are treated as detailed in this specification and also result in the termination of the control channel and the associated SIP dialog (issue a BYE request).
- The receipt of a 200 response to a SYNCH message implies that the SIP dialog and control connection have been successfully correlated. The control channel can now be used for further interactions.

It should be noted that SYNCH messages can be sent at any point while the Control Channel is open from either side, once the initial exchange is complete. It should also be noted that if present, the contents of the "Keep-Alive" and "Dialog-ID" headers should not change and new values have no relevance as they are both negotiated for the lifetime of the session.

Once a successful control channel has been established, as defined in Section 4 and Section 5 (and the connection has been correlated, as described in previous paragraph), the two entities are now in a position to exchange relevant control framework messages. The remainder of this section provides details of the core set of methods and responses that MUST be supported for the core control framework. Future extensions to this document MAY define new methods and responses.

6.1. Constructing Requests

An entity acting as a Control Client is now able to construct and send new requests on a control channel and MUST adhere to the syntax defined in Section 9 (Note: either client 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 Section 4 and Section 5 of this document. A Control Client MUST create a unique control message 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 "SCFW" token for the purpose of easily extracting the transaction identifier. The transaction identifier MUST be globally unique over space and time. All required mandatory

and optional control framework headers are then inserted into the control message 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 SYNCH control message that is discussed in this section along with the mechanism from Section 6.1.4).

Any framework message that contains an associated payload MUST also include a 'Content-Length' and 'Content-Type' message header which represents the size of the message body in decimal number of octets. If no associated payload is to be added to the message, a 'Content-Length' header with a value of '0' is considered the same as one not being present.

When all of the headers have been included in the framework message, it is sent down the control channel established in Section 4.

It is a requirement that a Server receiving such a request respond quickly with an appropriate response (as defined in Section 6.2). A Control Client entity needs to wait for "Transaction-Time" time for a response before considering the transaction a failure.

[Editors Note:DP1 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

6.1.1. Sending CONTROL

A 'CONTROL' message is used by Control Client to invoke control commands on a Control Server. The message is constructed in the same way as any standard Control Framework message, as discussed previously 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. These specifications MUST conform to the format defined in Section 8.4. A CONTROL message containing a payload MUST include a 'Content-Type' header indicating the payload type defined by the control package.

6.1.2. Sending REPORT

A 'REPORT' message is used by a Control Server when processing of a CONTROL Command extends beyond a 'Transaction-Timeout'. In this case a 202 response is returned. Status updates and the final results of the command are then returned in subsequent REPORT messages. The extended reporting mechanism defined in Section 6.1.2.1 can be used for a wide variety of functions including long lived event reporting

associated with a transaction.

[Editors Note:DP2 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

All REPORT messages MUST contain the same transaction ID in the request start line that was present in the original CONTROL transaction. This allows both extended transactions and event notifications to be correlated with the original CONTROL transaction. A REPORT message containing a payload MUST include a 'Content-Length and 'Content-Type' header indicating the payload type defined by the control package and its length.

6.1.2.1. 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 command completed within that time, a 200 response code would have been sent. If the command did not complete within that time, the response code 202 would have been 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.

[Editors Note:DP3 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

A Control Server issuing a 202 response MUST contain a 'Timeout' message header. This header will contain a value in delta seconds that represents 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 (no corresponding REPORT message arrived).

The initial REPORT message MUST contain a 'Seq' (Sequence) message header with a value equal to '1' (It should be noted that 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 delta seconds that represents 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 'pending' or 'update', the Control Client MUST reset the timer for the associated extended CONTROL transaction to the indicated timeout period. If the timeout period approaches with no intended REPORT

messages being 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 'pending'. 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 of "Upper-limit" is 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 a timeout period of 10 seconds would be refreshed after 8 seconds.

[Editors Note:DP4 - Need to pick a time for "Upper-Limit" - Work Group input requested.]

Subsequent REPORT messages that provide additional information relating to the extended CONTROL transaction MUST also include and increment by 1 the 'Seq' header value. They 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 9.5. 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. A Control Framework UAC can then clean up any pending state associated with the original control transaction.

6.1.3. Control Channel Keep-Alive

It is reasonable to expect this document to be used in various network architectures. This will include 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 (NAT) and Firewalls. It is important, therefore, that this document provides a 'keep-alive' mechanism that enables the control channel being created to firstly be kept active during times of inactivity (most Firewalls have a timeout period after which connections are closed) and also provide

the ability for application level failure detection. It should be noted at this point that the following procedures apply explicitly to the control channel being created and for details relating to a SIP keep-alive mechanism implementers should seek guidance from SIP Outbound [11]. The following 'keep-alive' procedures SHOULD be implemented by all entities unless it can be guaranteed that deployments will only occur with entities in a co-located domain. It should be noted that choosing to not implement the 'keep-alive' mechanism in this section, even when in a co-located architecture, will reduce the ability to detect application level errors - especially during long periods of in-activity.

6.1.3.1. Timeout Negotiation

During the creation of the initial SYNCH primitive, the clients will also negotiate a timeout period for the control channel 'keep-alive' mechanism. The following rules SHOULD be obeyed:

- o If the Client initiating the SDP "Offer" has a COMEDIA 'setup' attribute equal to 'active', the 'k-alive' header MUST be included in the SYNCH message generated by the offerer. The value of the 'K-Alive' header SHOULD be in the range of 95 and 120 seconds (this is consistent with SIP Outbound[11]). The client that generated the SDP "Answer" ('passive' client) MUST copy the 'K-alive' header into the 200 response to the SYNCH message with the same value.
- o If the Client initiating the SDP "Offer" has a COMEDIA 'setup' attribute equal to 'passive', the 'K-alive' header parameter MUST be included in the SYNCH message generated by the answerer. The value of the 'K-alive' header SHOULD be in the range of 95 and 120 seconds. The client that generated the SDP "Offer" ('passive' client) MUST copy the 'K-alive' header into the 200 response to the SYNCH message with the same value.
- o If the Client initiating the SDP "Offer" has a COMEDIA 'setup' attribute equal to 'actpass', the 'K-Alive' header parameter MUST be included in the SYNCH 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 SYNCH 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 COMDEDIA 'setup' attribute, the original 'Offerer' will generate the SYNCH request and include the 'Keep-Alive' header. The value SHOULD be in the range 95 to 120 seconds.
- o Once negotiated, the keep-alive applies for the remainder of the Control Framework session. Any subsequent SYNCH 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 SYNCH messages as it has no meaning. If it is present it MUST be ignored.

- The 'K-alive' header MUST NOT be included when the COMEDIA 'setup' attribute is equal to 'holdconn'.
- [Editors Note:DP5 - holdconn needs more thought.]
- Following the previous steps ensures 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' Control Framework level messages. The following section describes in more detail how to generate the Control Framework 'K-ALIVE' message.

6.1.3.2. Generating Keep-Alive Messages

Once the SIP dialog has been established using the SDP 'Offer/Answer' mechanism and the underlying control channel has been established (including the initial identity handshake using SYNCH as discussed in Section 6), both the 'active' and 'passive' (as defined in COMEDIA[6]) clients MUST start a keep-alive timer equal to the value negotiated during the control channel SYNCH request/response exchange (the value from the 'k-alive' header in delta seconds).

When acting as an 'active' entity, a 'K-ALIVE' Control Framework message MUST be generated before the local 'keep-alive' timer fires. An active entity is free to send the K-ALIVE Control Framework message when ever it chooses. A guideline of 80% of the local 'keep-alive' timer is suggested. The 'passive' entity MUST generate a 200 OK Control Framework response to the K-ALIVE message and reset the local 'keep-alive' timer. No other Control Framework response is valid. On receiving the 200 OK Control Framework message, the 'active' entity MUST reset the local 'keep-alive' timer. If no 200 OK response is received to the K-ALIVE Control Framework message, before the local 'keep-alive' timer fires, the 'active' entity SHOULD tear down the SIP dialog and recover the associated control channel resources. The 'active' entity MAY choose to try and recover the connection by renegotiation using COMEDIA. It should be noted that the local 'active' keep-alive timer MUST be reset on receipt of any Control Framework message (request or response) from the passive entity.

When acting as a 'passive' entity, a 'K-ALIVE' Control Framework message MUST be received before the local 'keep-alive' timer fires. The 'passive' entity MUST generate a 200 OK control framework response to the K-ALIVE Control Framework message. On sending the 200 OK response, the 'passive' entity MUST reset the local 'keep-alive' timer. If no K-ALIVE message is received before the local 'keep-alive' timer fires, the 'passive' entity SHOULD tear down the SIP dialog and recover the associated control channel resources. The

'active' entity MAY try to and recover the connection by renegotiating using COMEDIA. It should be noted that the local 'passive' keep-alive timer MUST be reset on receipt of any Control Framework message (request or response) from the active entity.

6.1.4. Package Negotiation

As part of the SYNCH message exchange a client generating the request MUST include a "Packages" header, as defined in Section 9. The "Packages " header will contain a list of all Control Framework packages that can be supported within this control session (from the perspective of the entity creating the SYNCH message). All tokens MUST be SIP Control Framework packages that adhere to the rules set out in Section 8. The initial SYNCH message MUST at least contain a single value.

An entity receiving the initial SYNCH request should carefully examine the contents of the "Packages" header. The entity responding with a 200 response to the SYNCH header will also populate the "Packages" header with supported Control Framework packages. This entry only contain packages that are listed in the received SYNCH request (either all or a subset). This forms a common set of Control Packages that are supported by both parties. Any Control Packages supported by the receiving entity that are not listed in the SYNCH message MAY be placed in the "Supported" header of the response. This is to provide a hint to the client generating the SYNCH message that the receiving entity also supports the listed Control Packages.

If no packages are supported by the entity receiving the SYNCH 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 SYNCH message based on the 422 response or consider the interaction as a failure. This would lead to termination of the associated SIP dialog by sending a SIP BYE request, as per RFC 3261 [2].

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

Any Control Framework commands relating to a Control Package that is no longer supported by the session are received after re-negotiation, the receiving entity SHOULD respond with a 420 response. An entity MAY choose to honor such commands for a limited period of time but this is implementation specific.

6.2. Constructing Responses

A Control Client or Server, on receiving a request, MUST generate a response within 'Transaction-Time'. The response MUST conform to the ABNF defined in Section 9. The first line of the response MUST contain the transaction identifier used in first line of the request, as defined in Section 6.1. Responses MUST NOT include the 'Status' or 'Timeout' message headers - if they are included they have no meaning or semantics.

[Editors Note:DP6 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

A Control Client or Server MUST then include a status code in the first line of the constructed response. A Control Framework request (like CONTROL) that has been understood, and either the relevant actions for the control command have completed or a control command error is detected, uses the 200 Control Framework status code as defined in Section 7.1. A 200 response MAY include message bodies. If a 200 response does contain a payload it MUST include Content-Length and Content-Type headers. A 200 is the only response defined in this specification that allows a message body to be included. A client receiving a 200 class response then considers the control command transaction completed. A Control Framework request (like CONTROL) that is received and understood but requires processing that extends beyond 'Transaction-Time' time will return a 202 status code in the response. This will be followed by an REPORT message(s) as defined in Section 6.1.2. A Control Package SHOULD explicitly define the circumstances under which either 200 or 202 with subsequent processing takes place.

[Editors Note:DP7 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

If a Control Client or Server encounters problems with either a Control Framework request (like REPORT or CONTROL), an appropriate error code should be used in the response, as listed in Section 7. The generation of a non 2xx class response code to either a Control Framework request (like CONTROL or REPORT) will indicate failure of the transaction, and all associated state and resources should 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.

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 200 code indicates the completion of a successful transaction.

7.2. 202 Response Code

The 202 response code indicates the completion of a successful transaction with additional information to be provided at a later time through the REPORT mechanism defined in Section 6.1.2.

7.3. 400 Response Code

The 400 response indicates that the request was syntactically incorrect.

7.4. 403 Response Code

The server understood the request, but is refusing to fulfill it. The request SHOULD NOT be repeated.

7.5. 405 Response Code

Method not allowed. The primitive is not supported.

7.6. 420 Response Code

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

7.7. 421 Response Code

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

7.8. 422 Response Code

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

7.9. 423 Response Code

Recipient already has a transaction with the same transaction ID.

7.10. 481 Response Code

The 481 response indicates that the transaction of the request does not exist.

7.11. 500 Response Code

The 500 response indicates that the recipient does not understand the request

8. Control Packages

"Control Packages" are intended to specify behavior that extends the the capability defined in this document. "Control Packages" are not allowed to weaken "MUST" and "SHOULD" strength statements that are detailed in this document. A "Control Package" may strengthen "SHOULD" to "MUST" if justified by the specific usage of the framework.

In addition to normal sections expected in a standards-track RFC and SIP extension documents, authors of "Control Packages" need to address each of the issues detailed in the following subsections. The following sections MUST be used as a template and included appropriately in all Control-Packages.

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 12. The package name MUST also register a version number for the package which is separated with a '/' symbol

e.g. package_name/1.0. This enables updates to the package to be registered where appropriate. An initial version of a package MUST start with the value '1.0'. Subsequent versions MUST increment this number if the same package name is to be used. The exact increment is left to the discretion of the package author.

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 should explicitly detail the control messages 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. The Control Package will make strong statements (MUST strength) if the XML schema defined in Section 16.1 in Appendix A is to be supported. If only part of the schema is required (for example just 'connection-id' or just conf-id), the Control Package will make equally strong (MUST strength) statements.

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 should indicate the location of detailed syntax definitions and semantics for the appropriate body types.

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 should indicate the location of detailed syntax definitions and semantics for the appropriate body types. 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.

8.6. Audit

[EDITORS NOTE: DP12 - Need to include audit template mechanism.]

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 [16]. The syntax in this section uses the Augmented Backus-Naur Form (ABNF) as defined in RFC2234 [17].

```
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  = pSCFW SP transact-id SP method CRLF
control-resp-start = pSCFW SP transact-id SP status-code [SP comment] CRLF
comment = utf8text

pSCFW = %x53.43.46.57; SCFW in caps
transact-id = alpha-num-token
method = mCONTROL / mREPORT / mSYNCH / 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
mSYNCH = %x53.59.4E.43.48; SYNCH in caps
mK-ALIVE = %x4B.2D.41.4C.49.56.45;K-ALIVE in caps

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

headers = header-name CRLF

header-name = (Content-Length
 /Control-Package
 /Status
 /Seq
 /Timeout
 /Dialog-id
 /Packages
 /Supported
```

```
 /Keep-alive
 /ext-header) CRLF

Content-Length = "Content-Length:" SP 1*DIGIT
Control-Package = "Control-Package:" SP 1*alpha-num-token
Status = "Status:" SP ("pending" / "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 supported *(COMMA supported)
Keep-alive = "Keep-Alive:" SP delta-seconds

dialog-id-string = alpha-num-token "~" alpha-num-token ["~" alpha-num-token]
package-name = alpha-num-token
supported = alpha-num-token
delta-seconds = 1*DIGIT

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

control-content = Content-Type 2CRLF data CRLF

Content-Type = "Content-Type:" SP media-type
media-type = type "/" subtype *( ";" gen-param )
type = token
subtype = token

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)
           ; token is compared case-insensitive

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

data = *OCTET
ext-header = hname ":" SP hval CRLF

hname = ALPHA *token
```

```
hval = utf8text

utf8text = *(HTAB / %x20-7E / 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
```

The following table details a summary of the headers that can be contained in Control Framework interactions. The "where" columns details where headers can be used:

R: header field may only appear in requests;

r: header field may only appear in responses;

Blank indicates the header field may appear in either requests or responses.

2xx, 4xx, etc.: A numerical value or range indicates response codes with which the header field can be used;

An empty entry in the "where" column indicates that the header field may be present in all requests and responses.

The remaining columns list the specified methods and the presence of a specific header:

m: The header field is mandatory.
o: The header field is optional.
-: The header field is not applicable (ignored if present).

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

Figure 10: Table 1

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, SYNCHs with the Control Server, and issues a CONTROL request that can't be completed within "transaction-timeout" seconds, so the Control Server returns a 202 response code to extend the trqansaction. The Control Server then follows with REPORTs until the requested action has been completed. The SIP dialog is then terminated.

[Editors Note:DP8 - Need to pick a time for "Transaction-Time" - Work Group input requested.]

```
   Control Client                               Control Server
          |                                            |
          |       (1) SIP INVITE                       |
          |  ----------------------------------------> |
          |                                            |
          |       (2) SIP 200                          |
          |  <----------------------------------------  |
          |                                            |
          |       (3) SIP ACK                          |
          |  ----------------------------------------> |
          |                                            |
          |==>=======================================>==|
          |         Control Channel Established        |
```

```
                 |==>=======================================>==|
                 |                                             |
                 |       (4) SYNCH                             |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (5) 200                               |
                 |  <---------------------------------------   |
                 |                                             |
                 |       (6) CONTROL                           |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (7) 202                               |
                 |  <---------------------------------------   |
                 |                                             |
                 |       (8) REPORT (pending)                  |
                 |  <----------------------------------------  |
                 |                                             |
                 |       (9) 200                               |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (10) REPORT (update)                  |
                 |  <----------------------------------------  |
                 |                                             |
                 |       (11) 200                              |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (12) REPORT (terminate)               |
                 |  <----------------------------------------  |
                 |                                             |
                 |       (13) 200                              |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (14) SIP BYE                          |
                 |  ---------------------------------------->  |
                 |                                             |
                 |       (15) SIP 200                          |
                 |  <---------------------------------------   |
                 |=============================================|
                 |         Control Channel 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@examplae.com>
From: <sip:control-client@example.com>;tag=8937498
Via: SIP/2.0/UDP control-client.example.com;branch=z9hG412345678
CSeq: 1 INVITE
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-client@pc1.example.com>
Content-Type: application/sdp
Cotent-Length: [..]

v=0
o=originator 2890844526 2890842808 IN IP4 controller.example,com
s=-
c=IN IP4 control-client.example.com
m=application 7575 TCP/SCFW
a=setup:active
a=connection:new
```

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 control-client.example.com;branch=z9hG412345678
CSeq: 1 INVITE
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-client@pc2.example.com>
Content-Type: application/sdp
Content-Length: [..]

v=0
o=originator 2890844526 2890842808 IN IP4 controller.example,com
s=-
c=IN IP4 control-server.example.com
m=application 7575 TCP/SCFW
a=setup:passive
a=connection:new
```

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): SYNCH.

```
SCFW 8djae7khauj SYNCH
Dialog-ID: 8937498~893jhoeihjr8392@example.com~023983774
K-alive: 100
```

```
Packages: msc-ivr-basic/1.0
```

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

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

6.   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): CONTROL.

```
SCFW 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.

```
SCFW i387yeiqyiq 202
Timeout: 10
```

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

```
SCFW i387yeiqyiq REPORT
Seq: 1
Status: pending
Timeout: 10
```

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

```
SCFW i387yeiqyiq 200
Seq: 1
```

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

```
SCFW 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.

```
SCFW i387yeiqyiq 200
Seq: 2
```

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

```
SCFW 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.

```
SCFW i387yeiqyiq 200
Seq: 3
```

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

```
BYE sip:control-client@pc2.example.com SIP/2.0
To: <sip:control-server@example.com>
From: <sip:control-client@example.com>;tag=8937498
Via: SIP/2.0/UDP control-client.example.com;branch=z9hG423456789
CSeq: 2 BYE
Call-ID: 893jhoeihjr8392@example.com
```

15. 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 control-client.example.com;branch=z9hG423456789
CSeq: 2 BYE
Call-ID: 893jhoeihjr8392@example.com
```

11. Security Considerations

SIP Control Framework needs to provide confidentiality and integrity for the messages it transfers. It also needs to provide assurances that the connected host is the host that it meant to connect to and that the connection has not been hijacked.

SIP Control Framework is designed to comply with the security-related requirements documented in the control prtoocol requirements document[8]. Specific security measures employed by the SIP Control Framework are summarized in the following subsections.

11.1. Session Establishment

SIP Control Framework sessions are established as media sessions described by SDP within the context of a SIP dialog. In order to ensure secure rendezvous between Control Framework clients and servers, the following are required:

- o The SIP implementation in Control Framework clients and servers MUST support digest authentication as specified in RFC3261 [2] and 'Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)[18].
- o The SIP implementation in Control Framework clients and servers SHOULD employ SIPS: URIs as specified in RFC3261 [2].

[EDITORS NOTE:DP9 - Sip identity - is this too strong?]

[EDITORS NOTE:DP10 - WHAT DO WE SAY ABOUT S/MIME????]

11.2. Transport Level Protection

When using only TCP connections, the SIP Control Framework security is weak. Although the SIP Control 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 is carried over the Control Framework channel. Clients and servers must be properly authenticated and the control

channel must permit the use of both confidentiality and integrity for the data. To ensure control channel protection, Control Framework clients and servers MUST support TLS and SHOULD utilize it by default unless alternative control channel protection is used or a protected environment is guaranteed. Alternative control channel protection MAY be used if desired (e.g.IPSEC).

TLS is used to authenticate devices and to provide integrity and confidentiality for the header fields being transported on the control chanel. SIP Control Framowork elements MUST implement TLS and MUST also implement the TLS ClientExtendedHello extended hello information for server name indication as described in [19]. A TLS cipher-suite of TLS_RSA_WITH_AES_128_CBC_SHA[2] MUST be supported (other cipher-suites MAY also be supported).

11.3. 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 specifically carry any specific mechanism to overcome such conflicts but will provide a summary of how it can be achieved.

It can be determined that access to resources and use of control channels relates to policy. It is implementation detail as to the level of policy that is adopted for use with specification. 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 either using SIP digest or 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 provide a control channel response code(403) to indicate to the issuer of a command that it is not permitted. It should be noted that additional policy requirements might be defined and applied in individual packages that specify a finer granularity for access to resources etc.

12. IANA Considerations

This specification instructs IANA to create a new registry for SIP Control Framework parameters. The SIP Control Framework Parameter registry is a container for sub-registries. This section further introduces sub-registries for SIP Control Framework packages, method names, status codes, header field names, port and transport protocol.

Additionally, Section 12.6 registers new parameters in existing IANA registries.

12.1. Control Packages Registration Information

This specification establishes the Control Packages sub-registry under Control Framework Packages. New parameters in this sub-registry must be published in an RFC (either as an IETF submission or RFC Editor submission).

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; it also demonstrates all three possible permutations of package type, contact, and reference.

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

Package Name	Contact	Reference
example1	[Boulton]	
example2	[Boulton]	[RFCXXX]
example3		[RFCXXX]

12.1.1. Control Package Registration Template

```
To: ietf-sip-control@iana.org
Subject: Registration of new SIP Control Framework package

Package Name:

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

Published Specification(s):

    (Control packages require a published RFC.).

Person & email address to contact for further information:
```

12.2. Control Framework Method Names

This specification establishes the Methods sub-registry under Control Framework Parameters and initiates its population as follows. New parameters in this sub-registry must be published in an RFC (either as an IETF submission or RFC Editor submission).

```
CONTROL - [RFCXXX]
REPORT  - [RFCXXX]
SYNCH   - [RFCXXX]
```

The following information MUST be provided in an RFC publication in

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

12.3. Control Framework Status Codes

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

```
Code [RFC Number]
```

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

- o The status code number.
- o The RFC number in which the method is registered.

12.4. Control Framework Header Fields

This specification establishes the header field-Field sub-registry under SIP Control Framework Parameters. New parameters in this sub-registry must be published in an RFC (either as an IETF submission or RFC Editor submission). Its initial population is defined as follows:

Control-Package - [RFCXXXX]
Status - [RFCXXXX]
Seq - [RFCXXXX]
Timeout - [RFCXXXX]
Dialog-id - [RFCXXXX]
Packages - [RFCXXXX]
Supported - [RFCXXXX]
Keep-alive - [RFCXXXX]

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

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

12.5. Control Framework Port

[Editors Note:DP11 - To be discussed].

12.6. SDP Transport Protocol

the SIP Control Framework defines the new SDP protocol field values 'TCP/SCFW', 'TCP/TLS/SCFW', 'SCTP/SCFW' and 'SCTP/ TLS/SCFW", which should be registered in the sdp-parameters registry under "proto". The values have the following meaning:

- o TCP/SCFW: Indicates the SIP Control Framework when TCP is used as an underlying transport for the control channel.
- o TCP/TLS/SCFW: Indicates the SIP Control Framework when TLS over TCP is used as an underlying transport for the control channel.
- o SCTP/SCFW: Indicates the SIP Control Framework when SCTP is used as an underlying transport for the control channel.
- o SCTP/TLS/SCFW: Indicates the SIP Control Framework when TLS over SCTP is used as an underlying transport for the control channel.

Specifications defining new protocol values must define the rules for

the associated media format namespace. The 'TCP/SCFW', 'TCP/TLS/SCFW', 'SCTP/SCFW' and 'SCTP/TLS/SCFW' protocol values allow only one value in the format field (fmt), which is a single occurrence of "*". Actual format determination is made using the control package extension specific payloads.

13. Changes

Note to RFC Editor: Please remove this whole section.

13.1. Changes from 00 Version

- o Aligned tokens to be 'SCFW' (removed ESCS).
- o Content-Length not mandatory for messages with no payload.
- o Corrected changes to call flows from legacy versions.
- o Use of term 'Active UA' in section 7 + others.
- o Added 'notify' to status header of ABNF.
- o Changed 481 to be transaction specific.
- o Added '423' duplicate transaction ID response.
- o Added '405' method not allowed.
- o Added IANA section.
- o Added Security Considerations section (used MSRP and MRCPv2 as a template).
- o Removed noisy initial REPORT message - *Lorenzo please check text*.
- o Fixed ABNF - PLEASE CHECK.
- o Removed separate event mechanism and now all tied to CONTROL transaction (extended).
- o General scrub of text.
- o Organised 'Editors Notes' for discussion on the mailing list.

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 and Michael Bardzinski of Ubiquity Software, Adnan Saleem of Convedia, 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.

16. Appendix A

During the creation of the Control Framework it has become clear that there are number of components that are common across multiple packages. It has become apparent that it would be useful to collect such re-usable components in a central location. In the short term this appendix provides the place holder 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.

16.1. 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 'connection-id' attribute MUST be constructed by concatenating the 'Local' and 'Remote' SIP dialog identifier tags as defined in RFC3261 [2]. 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 'connection-id' attribute for a Control Framework command would be:

```
7HDY839~HJKSkyHS
```

If a session description has more than one media description (as identified by 'm=' in [9]) it is possible to explicitly reference them individually. When constructing the 'connection-id' attribute for a command that applies to a specific media ('m=') in an SDP description, an optional third component can be concatenated to the Connection reference key. It is again separated using the '~' character and uses the 'label' attribute as specified in [10]. So the format would be:

```
'Local Dialog tag' + '~' + 'Remote Dialog tag' + '~' + 'Label Attribute'
```

As an example, for an entity that has a SIP Local dialog identifier of '7HDY839', a Remote dialog identifier of 'HJKSkyHS' and an SDP label attribute of 'HUwkuh7ns', the 'connection-id' attribute for a Control Framework command would be:

```
7HDY839~HJKSkyHS~HUwkuh7ns
```

It should be noted that Control Framework requests initiated in conjunction with a SIP dialog will produce a different 'connection-id' 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 also 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 'conf' 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 'conf-id' attribute does not need to be constructed based on the overlying SIP dialog. The 'conf-id' attribute value is system specific and should be selected with relevant context and uniqueness.

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:include schemaLocation="common-schema.xsd"/-->

       <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>
```

17. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement

Levels", BCP 14, RFC 2119, March 1997.

[2] 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.

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

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

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

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

[7] Groves, C., Pantaleo, M., Anderson, T., and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[8] Dolly, M. and R. Even, "Media Server Control Protocol Requirements", draft-dolly-mediactrl-requirements-00 (work in progress), June 2007.

[9] Handley, M., "SDP: Session Description Protocol", draft-ietf-mmusic-sdp-new-26 (work in progress), January 2006.

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

[11] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-11 (work in progress), November 2007.

[12] 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.

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

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

[15] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson,

"RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

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

[17] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[18] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

[19] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[20] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

Authors' Addresses

Chris Boulton
Avaya
Building 3
Wern Fawr Lane
St Mellons
Cardiff, South Wales CF3 5EA

Email: cboulton@avaya.com

Tim Melanchuk
Rain Willow Communications

Email: tim.melanchuk@gmail.com

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

Email: scott.mcglashan@hp.com

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).