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 tightly associated external Servers, for example 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 XCON work group of 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 .................................................. 4
2. Conventions and Terminology ................................. 4
3. Overview ....................................................... 5
4. Locating External Server Resources .......................... 9
5. Controlling UAC Behavior - Control Channel Setup ........ 10
   5.1. Controlling UAC Behavior - Media Dialogs .............. 11
6. External Server UAS Behavior - Control Channel Setup ...... 12
7. Control Framework Interactions .............................. 13
   7.1. Constructing Requests ................................ 15
   7.1.1. Sending CONTROL .................................. 15
   7.1.2. Sending REPORT .................................. 16
   7.2. Constructing Responses ................................. 17
8. Response Code Descriptions ................................ 18
   8.1. 200 Response Code .................................... 18
   8.2. 202 Response Code .................................... 18
   8.3. 400 Response Code .................................... 18
   8.4. 403 Response Code .................................... 18
   8.5. 481 Response Code .................................... 18
   8.6. 500 Response Code .................................... 18
9. Control Packages ................................................ 18
   9.1. Control Package Name .................................. 19
   9.2. Framework Message Usage ................................. 19
   9.3. Common XML Support .................................... 19
   9.4. CONTROL Message Bodies ................................. 19
   9.5. REPORT Message Bodies ................................ 20
   9.6. Examples ................................................ 20
10. Network Address Translation (NAT) .......................... 20
11. Formal Syntax ................................................ 20
   11.1. SIP Formal Syntax .................................... 20
   11.2. Control Framework Formal Syntax ...................... 20
12. Common XML Component Definitions ......................... 23
13. Examples ....................................................... 25
14. Security Considerations .................................... 29
1. Introduction

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 Control Protocol Framework' 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 9.

Current IETF transport 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 [11] 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:
Media Server : A Media Server is an entity that performs media processing on behalf of a requesting agent or Media Control Client. In particular, a Media Server offers mixing, announcement, tone detection and generation, and object play and record services. The Media Server has a direct RTP relationship with the source or sink of the media flow.
Control Client : A Control Client is an entity that requests processing from an external 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. In the document, we often refer to this entity simply as "the Client".

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 voice 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:-
- Service location - Use SIP Proxies or Back-to-Back User Agents for discovering external 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, audit parameters, and so forth.
- 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[12] and Callee Capabilities[13]. The ability to select an external 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 external Server resources using these techniques is outlined in Section 5 of this document.

Figure 1: Basic Architecture

The example from Figure 1 conveys a 1:1 connection between the Control Client and the external Server. It is possible, if required, for multiple connections using separate SIP dialogs to be established between the Control Client and the external Server entities. Any of the connections created between the two entities can then be used for external Server control interactions. The control connections are agnostic to the overlying media sessions, and specific session information can be incorporated in the control interaction commands represented using the defined XML schema (as defined in this document and utilised in external control packages). The ability to have multiple connections allows for stronger redundancy and the ability to manage high volumes of traffic in busy systems.

[Editors Note: Still under discussion. How does an app server know, when there are multiple external servers, which specific server has any given media session? Next version of the draft will discuss the correlation procedures. The App server needs a control channel with the media server and needs to know which channel to use once the media session has been established. Sounds like a GRUU usage?]

Consider the following simple example for session establishment between a Client and an external 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 SIP INVITE request to the external
Server. The request contains the SIP option tag "escs" in a SIP "Require" header for the purpose of forcing the use of the mechanism described in this document. The SDP payload includes the required information for control channel negotiation. The COMEDIA [6] specification for setting up and maintaining reliable connections is used (more detail available in later sections).

The client MUST include details of control packages that are supported and, more specifically, that will be used within the control channel created. This is achieved through the inclusion of a SIP "Control-Packages" header. The "Control-Packages" header is defined and described later in this document.

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
Require: escs
Control-Packages: <example-package>
Call-ID: 7823987HJHG6
Content-Type: application/sdp

v=0
c=originator 2890844526 2890842808 IN IP4 controller.example.com
s=--
c=IN IP4 controller.example.com
m=application 7575 TCP/ESCS
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:
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) TCP connection to the Media 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 requests and responses. If required, after the control channel has been setup, media sessions can be established using standard SIP third party call control.

[Editors Note: See previous note: this is where we may need to mention how an App Server knows which external Server is responsible for any given media session.]

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.
(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 external Server (B2BUA type functionality). Using the interaction illustrated by (2), the User Agent is able to negotiate media capabilities with the external server using standard SIP mechanisms as defined in RFC 3261 [2] and RFC 3264 [5].

If not present in the SDP received by the Control Client from the User Agent(2), a media label SDP attribute, which is defined in [10], should be inserted for every media description (identified as m= line as defined in [9]). This provides flexibility for the Control Client, because it can generate control messages that specify a particular Media stream (between User Agent and external Server) within a SIP media dialog. If a Media label is not included in the Control XML command, it applies to all media associated with the dialog.

A non 2xx class SIP response received for the INVITE request indicates that no SIP dialog has been created and is treated as specified RFC 3261 [2]. One exception to this is the "496" (TODO: need to pick an appropriate response code) response code whose operation is defined in Section 6

4. Locating External Server Resources

Section will describe mechanisms for locating an external Server.
5. Controlling UAC Behavior - Control Channel Setup

On creating a new SIP INVITE request, a UAC can insist on using the mechanisms defined in this document. This is achieved by inserting a SIP Require header containing the option tag 'escs'. A SIP Require header with the value 'escs' MUST NOT be present in any other SIP request type.

If on creating a new SIP INVITE request, a UAC does not want to insist on the usage of the mechanisms defined in this document but merely that it supports them, a SIP Supported header MUST be included in the request with the option tag 'escs'.

The SIP INVITE MUST include a SIP "Control-Packages" header which MUST contain at least one valid entry and can contain multiple control packages if required.

If a reliable response is received (as defined RFC 3261 [2] and RFC 3262 [3]) that contains a SIP Require header containing the option tag 'escs', the mechanisms defined in this document are applicable to the newly created dialog.

Before the UAC can send a request, it MUST include a valid session description 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.

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

\[\text{c=<nettype> <addrtype> <connection-address>}\]

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:

\[\text{c=IN IP4 controller.example.com}\]

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

\[\text{m=<media> <port> <proto>}\]
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 the value
"TCP/ESCS" as defined in Section 15.3.2 of this document.

[Editors note: Need to cover other protocols so not TCP specific]

The SDP MUST also contain a number of SDP media attributes(a=) that
are specifically defined in the COMEDIA 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.

5.1. Controlling 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 12.1.

Similarly, the ability to identify and apply commands to a group of media dialogs 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 12.1.

Support for both the common attributes described here is specified as part of each Control Package definition, as detailed in Section 9.

6. External Server 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 the presence of the SIP Supported and Require headers containing the option tag ‘escs’. 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] for responding to SIP supported and Require headers. If support is conveyed in a reliable SIP provisional response, the mechanisms in RFC 3262 [3] MUST also be used.

When constructing a SIP success response, the SDP payload MUST be constructed using the semantics(Connection, Media and attribute) defined in Section 5 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 5 would look as illustrated below:

```plaintext
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 connection is now available to exchange commands, as defined in "Control Packages" and described in Section 9. The state of the SIP Dialog and the associated Control channel are now explicitly linked. If either party wishes to terminate a Control channel is 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 contained in a SIP Supported or Require header it MUST respond as detailed in RFC 3261 [2]. If the UAS does support the SIP extension contained in a SIP Require or Supported header but does not support one or more of the Control packages, as represented in the "Control-Packages" SIP header, it MUST respond with a SIP "496 Unknown Control Package" response code. The error response MUST conform to RFC 3261 [2] and MUST also include a "Control-Packages" SIP header which lists the control packages from the request that the UAS does not support. This provides the Controlling UAC with an explicit reason for failure and allows for re-submission of the request without the un-supported control package.

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 in the ‘Supported’ message header. For this extension a value of ‘escs’ MUST be included. Additionally, a SIP entity MUST include all the additional control packages that are associated with the Control channel. This is achieved by including a ‘Control-Packages’ SIP message header listing all relevant supported Control package tokens.

7. 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)
client 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.

- Once connected, the client 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 11.2 and MUST only contain one message header, ‘dialog-id’, which contains the SIP dialog information.

- The ‘dialog-id’ message header is constructed by concatenating the Local-tag, Call-ID and Remote-tag (as defined in Section 11.2) from the SIP dialog and separating with a ‘~’. See syntax defined in Section 12.1 and examples in Section 9.6. 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’.

- The client who initiated the connection MUST then send the SYNCH request. It should then wait for a period of 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 client 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.

Once a successful control channel has been established, as defined in Section 5 and Section 6 (and the connection has been correlated, as described in previous paragraph), the two entities are now in a position to exchange relevant control framework commands. The remainder of this section provides details of the core set of commands and responses that MUST be supported for the core control framework. Future extensions to this document MAY define new commands and responses.
7.1. Constructing Requests

An entity acting as a controlling UAC is now able to construct and send new requests on a control channel and MUST adhere to the syntax defined in Section 11. A request MUST also adhere to the syntax defined by the Control Packages negotiated in Section 5 and Section 6 of this document. A Control Client MUST create a unique transaction and associated identifier per request transaction. The transaction identifier is then included in the first line of a control framework request along with the method type (as defined in the ABNF in Section 11). 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 containing the value of the Control Package to which this specific request applies (Multiple packages can be negotiated per control channel).

Any Control Framework message constructed that contains an associated payload MUST also include a ‘Content-Length’ 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 CONTROL message, a ‘Content-Length’ header with a value of ’0’ MUST be included.

When all of the properties have been included in the Control Framework message, it is sent down the control channel established in Section 5.

It is a requirement that a Control Framework UAS receiving such a request respond immediately with an appropriate response (as discussed in Section 7.2). A Control Client entity needs to wait for an arbitrary amount of time for a response before considering the transaction a failure. A wait time of 15 seconds is RECOMMENDED.

7.1.1. Sending CONTROL

A ‘CONTROL’ message is used by an entity acting as a UAC Control Client to invoke control commands on an entity acting as a UAS Control Client. The message is constructed in the same way as any standard Control Framework message, as discussed in Section 7.1 and defined in Section 11. A CONTROL message MAY contain a message body. The explicit detail of the message payload contained in a CONTROL message is declared in the individual Control Package, as specified by this framework (defined in Section 9.4).
7.1.2. Sending REPORT

On receiving a CONTROL command, an entity acting as a Control Framework UAS MUST respond immediately with a status code for the request, as specified in Section 7.2. The response code 202 indicates that although the Control Framework transaction has been understood and completed, the requested command is still being processed. The REPORT message is used to update the status of the command request.

A Control Framework UAS entity issuing a 202 response MUST immediately issue a REPORT message that contains the same transaction ID in the request start line that was present in the original CONTROL transaction. The initial REPORT message MUST also contain a ‘Seq’ (Sequence) message header with a value equal to ‘1’ (It should be noted that the ‘Seq’ numbers at both Controlling UAC client and UAS for framework messages are independent). The initial REPORT message MUST also contain a ‘Status’ message header with a value of ‘pending’. This initial REPORT message MUST NOT contain a message body and is primarily used to establish a subsequent message transaction based on the initial CONTROL command. All REPORT messages for a particular 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 entire CONTROL transaction and associated state. On receiving a REPORT message, the Control Framework UAC MUST reset the counter to the indicated timeout period. This is then repeated for every REPORT message received for the associated CONTROL transaction (as indicated by the unique transaction ID). 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 CONTROL transaction, because no message body or semantics are permitted. It is RECOMMENDED that a minimum value of 10 and a maximum of ?? is used for the value of the ‘Timeout’ message header. It is also RECOMMENDED that a Control Framework UAS 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.

Subsequent REPORT messages that provide additional information relating to the original CONTROL command MUST also include and increment by 1 the ‘Seq’ header value. They MUST also include a ‘Status’ header with a value of ‘update’. An interim REPORT message
sent to update the CONTROL command status MAY contain a message body, as defined by individual Control Packages and specified in Section 9.5. A REPORT message sent updating the 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 request being reset to the interval contained in the ‘Timeout’ message header.

When all processing for a CONTROL command has taken place, the entity acting as a Control Framework UAS 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.

7.2. Constructing Responses

A Control Framework entity, on receiving a request, will be required to immediately generate a response. A Control Framework response MUST be generated and sent immediately and MUST conform to the ABNF defined in Section 11. The first line of the response message MUST contain the transaction identifier used in first line of the request, as defined in Section 7.1. Responses MUST NOT include the ‘Status’ or ‘Timeout’ message headers - if they are included they have no meaning or semantics. A 200 response MAY include message bodies if the entity responding is able to provide the specified Control Package information without the request transaction timing out. If a 200 response does contain a payload it MUST include a Content-Length header. A 200 is the only response defined in this specification that allows a message body to be included.

A Control Framework entity MUST then include a status code in the first line of the constructed response. A CONTROL request that has been understood, and the relevant actions for the control transaction completed uses the 200 status code as defined in Section 8.1. A client receiving a 200 class response then considers the control command completed. A CONTROL request that is received and understood but requires further processing will return a 202 status code in the response. This will be followed immediately by an initial REPORT message as defined in Section 7.1.2. The specific Control Package will explicitly define the circumstances under which either 200 or 202 with subsequent processing takes place.

If the receiving Control Framework entity encounters problems with either a REPORT or CONTROL request, an appropriate error code should be used in the response, as listed in Section 8. The generation of a
non 2xx class response code to either a CONTROL or REPORT message will result in 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.

8. Response Code Descriptions

The following response codes are defined for transactional responses to commands defined in Section 7.1. All response codes in this section MUST be supported and can be used in response to both CONTROL and REPORT messages, the exception being that you MUST NOT generate a 202 response to a REPORT message.

8.1. 200 Response Code

The 200 code indicates the completion of a successful transaction.

8.2. 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 7.1.2.

8.3. 400 Response Code

The 400 response indicates that the request was syntactically incorrect.

8.4. 403 Response Code

The 400 response indicates that the requested operation is illegal.

8.5. 481 Response Code

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

8.6. 500 Response Code

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

9. 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.

9.1. 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 event tokens is contained in Section 15

9.2. 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.

[Editors Note: Need to examine text.]

9.3. 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 12.1 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.

9.4. 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 7 (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.

9.5. 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 7 (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.

9.6. 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.

10. Network Address Translation (NAT)

[Editors Note: This section will look at geographically distributed systems where NAT traversal might be an issue. It will look at both the SIP media dialog traversal and the control channel traversal.]

11. Formal Syntax

11.1. SIP Formal Syntax

The ABNF for the "Control-Packages" SIP header is as follows:

Control-Packages = "Control-Packages" HCOLON control-package-value *(COMMA control-package-value)

control-package-value = token

11.2. Control Framework Formal Syntax

The Control Framework interactions use the UTF-8 transformation format as defined in RFC3629 [15]. The syntax in this section uses the Augmented Backus-Naur Form (ABNF) as defined in RFC2234 [16].

control-req-or-resp = control-request / control-response
control-request = control-req-start headers [control-content]
control-response = control-resp-start headers
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 / 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

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

headers = Content-Length
/Control-Package
/Status
/Seq
/Dialog-id
/ext-header

Content-Length = "Content-Length:" SP 1*DIGIT
Control-Package = "Control-Package:" SP 1*alpha-num-token
Status = "Status:" SP ("pending" / "update" / "terminate")
Seq = "Seq:" SP 1*DIGIT
Dialog-id = "Dialog-id:" SP dialog-id-string

dialog-id-string = alpha-num-token "~" alpha-num-token ["~" alpha-num-token]

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)
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;

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.
12. Common XML Component Definitions

This section provides the XML schema definitions for the commonly used components that can be used in Control Packages.


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 Conference 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 conference. 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
<?xml version="1.0" encoding="UTF-8"?>
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:urn="urn:ietf:params:xml:ns:control:framework-attributes"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!-- xs: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="connection-id" type="xsd:string"/>
        <xsd:attribute name="conf-id" type="xsd:string"/>
    </xsd:attributeGroup>
</xsd:schema>
```
13. 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.

<table>
<thead>
<tr>
<th>Control Client</th>
<th>Control Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) SIP INVITE</td>
<td>&lt;----------------&gt;</td>
</tr>
<tr>
<td>(2) SIP 200</td>
<td>&lt;----------------&gt;</td>
</tr>
<tr>
<td>(3) SIP ACK</td>
<td>&lt;----------------&gt;</td>
</tr>
</tbody>
</table>

===>=======================================>==

Control Channel Established

===>=======================================>==

(4) SYNCH
<----------------> |
(5) 200
<----------------> |
(6) CONTROL
<----------------> |
(7) 202
<----------------> |
(8) REPORT (pending)
<----------------> |
(9) 200
<----------------> |
(10) REPORT (update)
<----------------> |
(11) 200
<----------------> |
(12) REPORT (terminate)
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 control-client.example.com;branch=z9hG412345678
   CSeq: 1 INVITE
   Require: escs
   Control-Packages: <example-package>
   Call-ID: 89jhoeihjr8392@example.com
   Contact: <sip:control-client@pc1.example.com>
   Content-Type: application/sdp

   v=0
   o=originator 2890844526 2890842808 IN IP4 controller.example.com
   s=-
   c=IN IP4 control-client.example.com
   m=application 7575 TCP/ESCS
   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
Require: escs
Control-Packages: <example-package>
Call-ID: 893jhoeihjr8392@example.com
Contact: <sip:control-client@pc2.example.com>
Content-Type: application/sdp

v=0
o=originator 2890844526 2890842808 IN IP4 controller.example,com
s=-
c=IN IP4 control-server.example.com
m=application 7575 TCP/ESCS
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


SCFW 8djae7khauj 200

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 i387yeiqyig CONTROL
Control-Package: <package-name>
Content-Length: 11

<XML BLOB/>
7. Control Server-->Control Client (Control Framework Message):
   202.
   
   SCFW i387yeiqyiq 202

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
    <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
    <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
   Require: escs
   Control-Packages: <example-package>
   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
   Require: escs
   Control-Packages: <example-package>
   Call-ID: 893jhoeihjr8392@example.com

14. Security Considerations

   Security Considerations to be included in later versions of this document.

15. IANA Considerations

15.1. IANA Registration of the ‘escs’ Option Tag

15.2. Control Package Registration Information

15.2.1. Control Package Registration Template

15.3. SDP Transport Protocol

15.3.1. TCP/ESCS

15.3.2. TCP/TLS/ESCS
15.4. SDP Attribute Names

15.5. SIP Response Codes

16. 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.

17. References

17.1. Normative References


17.2. Informative References


Authors’ Addresses

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

Email: cboulton@ubiquitysoftware.com

Tim Melanchuk
BlankSpace

Email: tim.melanchuk@gmail.com

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

Email: scott.mcglashan@hp.com

Asher Shiratzky
Radvision
24 Raoul Wallenberg st
Tel-Aviv, Israel

Email: ashers@radvision.com
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.