Protocol for Controlling Multiple Streams for Telepresence (CLUE)
draft-ietf-clue-protocol-19

Abstract

The CLUE protocol is an application protocol conceived for the description and negotiation of a telepresence session. The design of the CLUE protocol takes into account the requirements and the framework defined within the IETF CLUE working group. A companion document delves into CLUE signaling details, as well as on the SIP/SDP session establishment phase. CLUE messages flow over the CLUE data channel, based on reliable and ordered SCTP over DTLS transport. Message details, together with the behavior of CLUE Participants acting as Media Providers and/or Media Consumers, are herein discussed.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents.
carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
2. Terminology ............................................ 3
3. Conventions ............................................. 4
4. Overview of the CLUE protocol .......................... 4
5. Protocol messages ....................................... 7
  5.1. options ............................................ 9
  5.2. optionsResponse .................................... 11
  5.3. advertisement ..................................... 12
  5.4. ack ............................................... 13
  5.5. configure ........................................ 14
  5.6. configureResponse ................................. 15
  5.7. Response codes and reason strings .................. 16
6. Protocol state machines .................................. 18
  6.1. Media Provider’s state machine ..................... 21
  6.2. Media Consumer’s state machine .................... 22
7. Versioning ................................................ 25
8. Extensions ............................................... 25
  8.1. Extension example .................................. 27
9. XML Schema .............................................. 29
10. Call flow example ...................................... 34
  10.1. CLUE message nr. 1: ‘options’ ..................... 37
  10.2. CLUE message nr. 2: ‘optionsResponse’ .......... 39
  10.3. CLUE message nr. 3: ‘advertisement’ ............ 39
  10.4. CLUE message nr. 4: ‘configure + ack’ ........... 47
  10.5. CLUE message nr. 5: ‘confResponse’ ............... 47
  10.6. CLUE message nr. 6: ‘advertisement’ ............ 48
  10.7. CLUE message nr. 7: ‘ack’ ....................... 58
  10.8. CLUE message nr. 8: ‘configure’ .................. 58
  10.9. CLUE message nr. 9: ‘confResponse’ .............. 59
11. Security Considerations .................................. 60
12. IANA Considerations .................................... 61
  12.1. URN Sub-Namespace Registration .................. 61
  12.2. XML Schema registration .......................... 62
  12.3. MIME Media Type Registration for ‘application/clue+xml’ 62
  12.4. CLUE Protocol Registry ............................ 63
    12.4.1. CLUE Message Types .......................... 63
    12.4.2. CLUE Response Codes ........................ 64
13. Acknowledgments ....................................... 66
14. References ............................................. 66
  14.1. Normative References ............................. 66
14.2. Informative References .......................... 67
Authors’ Addresses ........................................ 67

1. Introduction

The CLUE protocol is an application protocol used by two CLUE Participants to enhance the experience of a multimedia telepresence session. The main goals of the CLUE protocol are:

1. enabling a Media Provider (MP) to properly announce its current telepresence capabilities to a Media Consumer (MC) in terms of available media captures, groups of encodings, simultaneity constraints and other information defined in [I-D.ietf-clue-framework];

2. enabling an MC to request the desired multimedia streams from the offering MP.

CLUE-capable endpoints are connected by means of the CLUE data channel, an SCTP over DTLS channel that is opened and established as described in [I-D.ietf-clue-signaling] and [I-D.ietf-clue-datachannel]. CLUE protocol messages flowing over such a channel are detailed in this document, both syntactically and semantically.

In Section 4 we provide a general overview of the CLUE protocol. CLUE protocol messages are detailed in Section 5. The CLUE Protocol state machines are introduced in Section 6. Versioning and extensions are discussed in Section 7 and Section 8, respectively. The XML schema defining the CLUE messages is reported in Section 9.

2. Terminology

This document refers to the same terminology used in [I-D.ietf-clue-framework] and in [RFC7262]. We briefly recall herein some of the main terms used in the document. The definition of "CLUE Participant" herein proposed is not imported from any of the above documents.

Capture Encoding: A specific encoding of a Media Capture, to be sent via RTP [RFC3550].

CLUE Participant (CP): An entity able to use the CLUE protocol within a telepresence session. It can be an endpoint or an MCU (Multipoint Control Unit) able to use the CLUE protocol.
CLUE-capable device: A device that supports the CLUE data channel [I-D.ietf-clue-datachannel], the CLUE protocol and the principles of CLUE negotiation, and seeks CLUE-enabled calls.

Endpoint: A CLUE-capable device that is the logical point of final termination through receiving, decoding and rendering, and/or initiation through capturing, encoding, and sending of media streams. An endpoint consists of one or more physical devices that source and sink media streams, and exactly one [RFC4353] Participant (which, in turn, includes exactly one [RFC3261] User Agent). Endpoints can be anything from multiscreen/multicamera rooms to handheld devices.

Multipoint Control Unit (MCU): a CLUE-capable device that connects two or more endpoints together into one single multimedia conference [RFC7667]. An MCU includes an [RFC4353]-like Mixer, without the [RFC4353] requirement to send media to each participant.

Media: Any data that, after suitable encoding, can be conveyed over RTP, including audio, video or timed text.

Media Capture: a source of Media, such as from one or more Capture Devices or constructed from other Media streams.

Media Consumer (MC): A CLUE Participant (i.e., an Endpoint or an MCU) able to receive Capture Encodings.

Media Provider (MP): A CLUE Participant (i.e., an Endpoint or an MCU) able to send Capture Encodings.

Stream: a Capture Encoding sent from a Media Provider to a Media Consumer via RTP [RFC3550].

3. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Overview of the CLUE protocol

The CLUE protocol is conceived to enable CLUE telepresence sessions. It is designed in order to address SDP limitations in terms of the description of some information about the multimedia streams that are involved in a real-time multimedia conference. Indeed, by simply
using SDP it is not possible to convey information about the features of the flowing multimedia streams that are needed to enable a "being there" rendering experience. Such information is contained in the CLUE framework document [I-D.ietf-clue-framework] and formally defined and described in the CLUE data model document [I-D.ietf-clue-data-model-schema]. The CLUE protocol represents the mechanism for the exchange of telepresence information between CLUE Participants. It mainly provides the messages to enable a Media Provider to advertise its telepresence capabilities and to enable a Media Consumer to select the desired telepresence options.

The CLUE protocol, as defined in the following, is a stateful, client-server, XML-based application protocol. CLUE protocol messages flow on a reliable and ordered SCTP over DTLS transport channel connecting two CLUE Participants. Messages carry information taken from the XML-based CLUE data model ([I-D.ietf-clue-data-model-schema]). Three main communication phases can be identified:

1. Establishment of the CLUE data channel: in this phase, the CLUE data channel setup takes place. If it completes successfully, the CPs are able to communicate and start the initiation phase.

2. Negotiation of the CLUE protocol version and extensions (initiation phase): the CPs connected via the CLUE data channel agree on the version and on the extensions to be used during the telepresence session. Special CLUE messages are used for such a task (‘options’ and ‘optionsResponse’). The version and extensions negotiation can be performed once during the CLUE session and only at this stage. At the end of that basic negotiation, each CP starts its activity as a CLUE MP and/or CLUE MC.

3. CLUE telepresence capabilities description and negotiation: in this phase, the MP-MC dialogues take place on the data channel by means of the CLUE protocol messages.

As soon as the channel is ready, the CLUE Participants must agree on the protocol version and extensions to be used within the telepresence session. CLUE protocol version numbers are characterized by a major version number (single digit) and a minor version number (single digit), both unsigned integers, separated by a dot. While minor version numbers denote backward compatible changes in the context of a given major version, different major version numbers generally indicate a lack of interoperability between the protocol implementations. In order to correctly establish a CLUE dialogue, the involved CPs must have in common a major version number (see Section 7 for further details). The subset of the extensions
that are allowed within the CLUE session is also determined in the
initiation phase. It includes only the extensions that are supported
by both parties. A mechanism for the negotiation of the CLUE
protocol version and extensions is part of the initiation phase.
According to such a solution, the CP that is the CLUE Channel
Initiator (CI) issues a proper CLUE message (‘options’) to the CP
that is the Channel Receiver (CR) specifying the supported version
and extensions. The CR then answers by selecting the subset of the
CI extensions that it is able to support and determines the protocol
version to be used.

After the negotiation phase is completed, CLUE Participants describe
and agree on the media flows to be exchanged. In many cases CPs will
seek to both transmit and receive media. Hence in a call between two
CPs, A and B, there would be two separate message exchange sequences,
as follows:

1. the one needed to describe and set up the media streams sent from
   A to B, i.e., the dialogue between A’s Media Provider side and
   B’s Media Consumer side

2. the one needed to describe and set up the media streams sent from
   B to A, i.e., the dialogue between B’s Media Provider side and
   A’s Media Consumer side

CLUE messages for the media session description and negotiation are
designed by considering the MP side as the server side of the
protocol, since it produces and provides media streams, and the MC
side as the client side of the protocol, since it requests and
receives media streams. The messages that are exchanged to set up
the telepresence media session are described by focusing on a single
MP-MC dialogue.

The MP first advertises its available media captures and encoding
capabilities to the MC, as well as its simultaneity constraints,
according to the information model defined in
[I-D.ietf-clue-framework]. The CLUE message conveying the MP’s
multimedia offer is the ‘advertisement’ message. Such message
leverages the XML data model definitions provided in
[I-D.ietf-clue-data-model-schema].

The MC selects the desired streams of the MP by using the ‘configure’
message, which makes reference to the information carried in the
previously received ‘advertisement’.

Besides ‘advertisement’ and ‘configure’, other messages have been
conceived in order to provide all the needed mechanisms and
operations. Such messages are detailed in the following sections.
5. Protocol messages

CLUE protocol messages are textual, XML-based messages that enable the configuration of the telepresence session. The formal definition of such messages is provided in the XML Schema provided at the end of this document (Section 9). This section includes non-normative excerpts of the schema to aid in describing it.

The XML definitions of the CLUE information provided in [I-D.ietf-clue-data-model-schema] are included within some CLUE protocol messages (namely the ‘advertisement’ and the ‘configure’ messages), in order to use the concepts defined in [I-D.ietf-clue-framework].

The CLUE protocol messages are the following:

- options
- optionsResponse
- advertisement
- ack
- configure
- configureResponse

While the ‘options’ and ‘optionsResponse’ messages are exchanged in the initiation phase between the CPs, the other messages are involved in MP-MC dialogues. Please note that the word "dialog" in this document is not related to SIP’s usage of the same term. It refers to message exchange sequences between a CLUE Media Provider and a Clue Media Consumer.

Each CLUE message inherits a basic structure depicted in the following excerpt (Figure 1):
The information contained in each CLUE message is:

- **clueId**: an optional XML element containing the identifier (in the form of a generic string) of the CP within the telepresence system;

- **sequenceNr**: an XML element containing the local message sequence number. The sender MUST increment the sequence numbers by one for each new message sent, the receiver MUST remember the most recent sequence number received and send back a 402 error if it receives a message with an unexpected sequence number (e.g., sequence number gap, repeated sequence number, sequence number too small). The initial sequence number can be chosen randomly by each party;

- **protocol**: a mandatory attribute set to "CLUE", identifying the protocol the messages refer to;

- **v**: a mandatory attribute carrying the version of the protocol. The content of the "v" attribute is composed by the major version number followed by a dot and then by the minor version number of the CLUE protocol in use. The major number cannot be "0" and, if it is more than one digit, it cannot start with a "0". Allowed values are of this kind are, e.g., "1.3", "2.0", "20.44" etc. This document describes version 1.0.

Each CP is responsible for creating and updating up to three independent streams of sequence numbers in messages it sends: (i) one for the messages sent in the initiation phase, (ii) one for the
messages sent as MP (if it is acting as a MP), and (iii) one for the messages sent as MC (if it is acting as a MC).

In particular, CLUE response messages ('optionsResponse', 'ack', 'configureResponse') derive from a base type, inheriting from the clueMessageType, which is defined as follows (Figure 2):

```xml
<xs:complexType name="clueResponseType">
  <xs:complexContent>
    <xs:extension base="clueMessageType">
      <xs:sequence>
        <xs:element name="responseCode" type="responseCodeType"/>
        <xs:element name="reasonString" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

*Figure 2: Structure of CLUE response messages*

The elements <responseCode> and <reasonString> get populated as detailed in Section 5.7

5.1. options

The ‘options’ message is sent by the CLUE Participant that is the Channel Initiator to the CLUE Participant that is the Channel Receiver as soon as the CLUE data channel is ready. Besides the information envisioned in the basic structure, it specifies:

- <mediaProvider>: a mandatory boolean field set to "true" if the CP is able to act as a MP
- <mediaConsumer>: a mandatory boolean field set to "true" if the CP is able to act as a MC
- <supportedVersions>: the list of the supported versions
- <supportedExtensions>: the list of the supported extensions

The XML Schema of such a message is reported below (Figure 3):

```xml
<!-- CLUE OPTIONS -->
<xs:complexType name="optionsMessageType">
  <xs:complexContent>
    <xs:extension base="clueMessageType">
      <xs:sequence>
        <xs:element name="mediaProvider" type="xs:boolean"/>
        <xs:element name="mediaConsumer" type="xs:boolean"/>
        <xs:element name="supportedVersions" type="xs:string" minOccurs="0"/>
        <xs:element name="supportedExtensions" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```
<xs:extension base="clueMessageType">
  <xs:sequence>
    <xs:element name="mediaProvider" type="xs:boolean" />
    <xs:element name="mediaConsumer" type="xs:boolean" />
    <xs:element name="supportedVersions" type="versionsListType" minOccurs="0" />
    <xs:element name="supportedExtensions" type="extensionsListType" minOccurs="0" />
    <xs:any namespace="##other" processContents="lax" minOccurs="0" />
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:extension>

<!-- VERSIONS LIST TYPE -->
<xs:complexType name="versionsListType">
  <xs:sequence>
    <xs:element name="version" type="versionType" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" />
</xs:complexType>

<!-- EXTENSIONS LIST TYPE -->
<xs:complexType name="extensionsListType">
  <xs:sequence>
    <xs:element name="extension" type="extensionType" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" />
</xs:complexType>

<!-- EXTENSION TYPE -->
<xs:complexType name="extensionType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="schemaRef" type="xs:anyURI" />
    <xs:element name="version" type="versionType" />
  </xs:sequence>
  <xs:any namespace="##other" processContents="lax" minOccurs="0" />
</xs:complexType>
Figure 3: Structure of CLUE 'options' message

<supportedVersions> contains the list of the versions that are supported by the CI, each one represented in a child <version> element. The content of each <version> element is a string made by the major version number followed by a dot and then by the minor version number (e.g., 1.3 or 2.4). Exactly one <version> element MUST be provided for each major version supported, containing the maximum minor version number of such a version, since all minor versions are backward compatible. If no <supportedVersions> is carried within the 'options' message, the CI supports only the version declared in the "v" attribute and all the versions having the same major version number and lower minor version number. For example, if the "v" attribute has a value of "3.4" and there is no <supportedVersions> tag in the 'options' message, it means the CI supports only major version 3 with all the minor versions comprised between 3.0 and 3.4, with version 3.4 included. If a <supportedVersion> is provided, at least one <version> tag MUST be included. In this case, the "v" attribute SHOULD be set to the largest minor version of the smallest major version advertised in the <supportedVersions> list. Indeed, the intention behind the "v" attribute is that some implementation that receives a version number in the "v" field with a major number higher than it understands is supposed to close the connection, since it runs a risk of misinterpreting the contents of messages. The minor version is less useful in this context, since minor versions are defined to be both backwards and forwards compatible, but it is more useful to know the highest minor version supported than some random minor version, as it indicates the full feature set that is supported. The reason why it is less useful is that the value can in any case be parsed out of the <version> list.

The <supportedExtensions> element specifies the list of extensions supported by the CI. If there is no <supportedExtensions> in the 'options' message, the CI does not support anything other than what is envisioned in the versions it supports. For each extension, an <extension> element is provided. An extension is characterized by a name, an XML schema of reference where the extension is defined, and the version of the protocol which the extension refers to.

5.2. optionsResponse

The 'optionsResponse' (Figure 4) is sent by a CR to a CI as a reply to the 'options' message. The 'optionsResponse' contains a mandatory response code and a reason string indicating the processing result of the 'options' message. If the responseCode is between 200 and 299 inclusive, the response MUST also include <mediaProvider>, <mediaConsumer>, <version> and <commonExtensions> elements; it MAY
include them for any other response code. `<mediaProvider>` and `<mediaConsumer>` elements (which are of a boolean nature) are associated with the supported roles (in terms of, respectively MP and MC), similarly to what the CI does in the ‘options’ message. The `<version>` field indicates the highest commonly supported version number. The content of the `<version>` element MUST be a string made of the major version number followed by a dot and then by the minor version number (e.g., 1.3 or 2.4). Finally, the commonly supported extensions are copied in the `<commonExtensions>` field.

```xml
<!-- CLUE ‘optionsResponse’ -->
<xs:complexType name="optionsResponseMessageType">
  <xs:complexContent>
    <xs:extension base="clueResponseType">
      <xs:sequence>
        <xs:element name="mediaProvider" type="xs:boolean" minOccurs="0"/>
        <xs:element name="mediaConsumer" type="xs:boolean" minOccurs="0"/>
        <xs:element name="version" type="versionType" minOccurs="0"/>
        <xs:element name="commonExtensions" type="extensionsListType" minOccurs="0"/>
      </xs:sequence>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Figure 4: Structure of CLUE ‘optionsResponse’ message

Upon reception of the ‘optionsResponse’ the version to be used is the one provided in the `<version>` tag of the message. The following CLUE messages MUST use such a version number in the "v" attribute. The allowed extensions in the CLUE dialogue are those indicated in the `<commonExtensions>` of the ‘optionsResponse’ message.

5.3. advertisement

The ‘advertisement’ message is used by each MP to advertise the available media captures and related information to the corresponding MC. The MP sends an ‘advertisement’ to the MC as soon as it is ready after the successful completion of the initiation phase, i.e., as soon as the version and the extensions of the CLUE protocol are agreed between the CPs. During a single CLUE session, an MP may send
new ‘advertisement’ messages to replace the previous advertisement, if, for instance, its CLUE telepresence media capabilities change mid-call. A new ‘advertisement’ completely replaces the previous ‘advertisement’.

The ‘advertisement’ structure is defined in the schema excerpt below (Figure 5). The ‘advertisement’ contains elements compliant with the CLUE data model that characterize the MP’s telepresence offer. Namely, such elements are: the list of the media captures (<mediaCaptures>), of the encoding groups (<encodingGroups>), of the capture scenes (<captureScenes>), of the simultaneous sets (<simultaneousSets>), of the global views (<globalViews>), and of the represented participants (<people>). Each of them is fully described in the CLUE framework document and formally defined in the CLUE data model document.

```xml
<!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
<xs:complexType name="advertisementMessageType">
  <xs:complexContent>
    <xs:extension base="clueMessageType">
      <xs:sequence>
        <!-- mandatory -->
        <xs:element name="mediaCaptures" type="dm:mediaCapturesType"/>
        <xs:element name="encodingGroups" type="dm:encodingGroupsType"/>
        <xs:element name="captureScenes" type="dm:captureScenesType"/>
        <!-- optional -->
        <xs:element name="simultaneousSets" type="dm:simultaneousSetsType" minOccurs="0"/>
        <xs:element name="globalViews" type="dm:globalViewsType" minOccurs="0"/>
        <xs:element name="people" type="dm:peopleType" minOccurs="0"/>
      </xs:sequence>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 5: Structure of CLUE ‘advertisement’ message

5.4.  ack

The ‘ack’ message is sent by a MC to a MP to acknowledge an ‘advertisement’ message. As it can be seen from the message schema provided in the following excerpt (Figure 6), the ‘ack’ contains a
response code and may contain a reason string for describing the processing result of the ‘advertisement’. The <advSequenceNr> carries the sequence number of ‘advertisement’ message the ‘ack’ refers to.

```xml
<!-- 'ack' MESSAGE TYPE -->
<xs:complexType name="advAcknowledgementMessageType">
  <xs:complexContent>
    <xs:extension base="clueResponseType">
      <xs:sequence>
        <xs:element name="advSequenceNr" type="xs:positiveInteger"/>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 6: Structure of CLUE ‘ack’ message

5.5. configure

The ‘configure’ message is sent from a MC to a MP to list the advertised captures the MC wants to receive. The MC MUST send a ‘configure’ after the reception of an ‘advertisement’, as well as each time it wants to request other captures that have been previously advertised by the MP. The content of the ‘configure’ message is shown below (Figure 7).
Figure 7: Structure of CLUE 'configure' message

The <advSequenceNr> element contains the sequence number of the 'advertisement' message the 'configure' refers to.

The optional <ack> element, when present, contains a success response code, as defined in Section 5.7. It indicates that the 'configure' message also acknowledges with success the referred advertisement ('configure' + 'ack' message). The <ack> element MUST NOT be present if an 'ack' message (associated with the advertisement carrying that specific sequence number) has been already sent back to the MP.

The most important content of the 'configure' message is the list of the capture encodings provided in the <captureEncodings> element (see [I-D.ietf-clue-data-model-schema] for the definition of <captureEncodings>). Such an element contains a sequence of capture encodings, representing the streams to be instantiated.

5.6. configureResponse

The 'configureResponse' message is sent from the MP to the MC to communicate the processing result of requests carried in the previously received 'configure' message. It contains (Figure 8) a response code (and optionally a reason string) indicating either the success or the failure (along with failure details) of a 'configure' request processing. Following, the <confSequenceNr> field contains the sequence number of the 'configure' message the response refers to. There is no partial execution of commands. As an example, if a
MP is able to understand all the selected capture encodings except one, then the whole command fails and nothing is instantiated.

```xml
<xs:complexType name="configureResponseMessageType">
  <xs:complexContent>
    <xs:extension base="clueResponseType">
      <xs:sequence>
        <xs:element name="confSequenceNr" type="xs:positiveInteger" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" />
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 8: Structure of CLUE ‘configureResponse’ message

5.7. Response codes and reason strings

Response codes are defined as a sequence of three digits. A well-defined meaning is associated with the first digit. Response codes beginning with "2" are associated with successful responses. Response codes that do not begin with either "2" or "1" indicate an error response, i.e., that an error occurred while processing a CLUE request. In particular, response codes beginning with "3" indicate problems with the XML content of the message ("Bad syntax", "Invalid value", etc.), while response codes beginning with "4" refer to problems related to CLUE protocol semantics ("Invalid sequencing", "Version not supported", etc.). 200, 300 and 400 codes are the most generic ones in their respective categories. Further response codes can be either defined in future versions of the protocol, or defined by leveraging the extension mechanism. In both cases, the new response codes MUST be registered with IANA. Such new response codes MUST NOT override the ones here defined and they MUST respect the semantics of the first code digit.

This document does not define response codes starting with "1", and such response codes are not allowed to appear in major version 1 of the CLUE protocol. The range from 100 to 199 inclusive is reserved for future major versions of the protocol to define response codes for delayed or incomplete operations if necessary. Response codes starting with "5" through "9" are reserved for future major versions of the protocol to define new classes of response, and are not
allowed in major version 1 of the CLUE protocol. Response codes starting with "0" are not allowed.

The response codes and strings defined for use with version 1 of the CLUE protocol are listed in Figure 9. The "Description" text contained in the table can be sent in the <reasonString> element of a response message. Implementations can (and are encouraged to) include more specific descriptions of the error condition, if possible.

<table>
<thead>
<tr>
<th>Response code</th>
<th>Reason string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
<td>The request has been successfully processed.</td>
</tr>
<tr>
<td>300</td>
<td>Low-level request error.</td>
<td>A generic low-level request error has occurred.</td>
</tr>
<tr>
<td>301</td>
<td>Bad syntax</td>
<td>The XML syntax of the message is not correct.</td>
</tr>
<tr>
<td>302</td>
<td>Invalid value</td>
<td>The message contains an invalid parameter value.</td>
</tr>
<tr>
<td>303</td>
<td>Conflicting values</td>
<td>The message contains values that cannot be used together.</td>
</tr>
<tr>
<td>400</td>
<td>Semantic errors</td>
<td>Semantic errors in the received CLUE protocol message.</td>
</tr>
</tbody>
</table>
### 6. Protocol state machines

The CLUE protocol is an application protocol used between two CPs in order to properly configure a multimedia telepresence session. CLUE protocol messages flow over the CLUE Data Channel, a DTLS/SCTP channel established as depicted in [I-D.ietf-clue-datachannel]. We herein discuss the state machines associated, respectively, with the CLUE Participant (Figure 10), with the MC role (Figure 11) and with the MP role (Figure 12). Endpoints often wish to both send and receive media, i.e., act as both MP and MC. As such there will often be two sets of messages flowing in opposite directions; the state machines of these two flows do not interact with each other. Only the CLUE application logic is considered. The interaction of CLUE protocol and SDP negotiations for the media streams exchanged is treated in [I-D.ietf-clue-signaling].

<table>
<thead>
<tr>
<th>Code</th>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Version not supported</td>
<td>The protocol version used in the message is not supported.</td>
</tr>
<tr>
<td>402</td>
<td>Invalid sequencing</td>
<td>Sequence number gap; repeated sequence number; sequence number outdated.</td>
</tr>
<tr>
<td>403</td>
<td>Invalid identifier</td>
<td>The clueId used in the message is not valid or unknown.</td>
</tr>
<tr>
<td>404</td>
<td>advertisement Expired</td>
<td>The sequence number of the advertisement the configure refers to is out of date.</td>
</tr>
<tr>
<td>405</td>
<td>Subset choice not allowed</td>
<td>The subset choice is not allowed for the specified Multiple Content Capture</td>
</tr>
</tbody>
</table>

Figure 9: CLUE response codes
The main state machines focus on the behavior of the CLUE Participant (CP) acting as a CLUE Channel Initiator/Receiver (CI/CR).

The initial state is the IDLE one. When in the IDLE state, the CLUE data channel is not established and no CLUE-controlled media are exchanged between the two considered CLUE-capable devices (if there is an ongoing exchange of media streams, such media streams are not currently CLUE-controlled).

When the CLUE data channel sets up ("start channel"), the CP moves from the IDLE state to the CHANNEL SETUP state.

If the CLUE data channel is successfully set up ("channel established"), the CP moves from the CHANNEL SETUP state to the OPTIONS state. Otherwise if "channel error", it moves back to the IDLE state. The same transition happens if the CLUE-enabled telepresence session ends ("session ends"), i.e., when an SDP negotiation for removing the CLUE data channel is performed.

When in the OPTIONS state, the CP addresses the initiation phase where both parts agree on the version and on the extensions to be used in the subsequent CLUE messages exchange phase. If the CP is the Channel Initiator (CI), it sends an ‘options’ message and waits for the ‘optionsResponse’ message. If the CP is the Channel Receiver (CR), it waits for the ‘options’ message and, as soon as it arrives, replies with the ‘optionsResponse’ message. If the negotiation is successfully completed ("OPTIONS phase success"), the CP moves from the OPTIONS state to the ACTIVE state. If the initiation phase fails ("OPTIONS phase failure"), the CP moves from the OPTIONS state to the IDLE state. The initiation phase might fail because of one of the following reasons:

1. the CI receives an ‘optionsResponse’ with an error response code
2. the CI does not receive any ‘optionsResponse’ and a timeout error is raised
3. the CR does not receive any ‘options’ and a timeout error is raised

When in the ACTIVE state, the CP starts the envisioned sub-state machines (i.e., the MP state machine and the MC state machine) according to the roles it plays in the telepresence sessions. Such roles have been previously declared in the ‘options’ and ‘optionsResponse’ messages involved in the initiation phase (see OPTIONS sections Section 5.1 and Section 5.2 for the details). When in the ACTIVE state, the CP delegates the sending and the processing of the CLUE messages to the appropriate MP/MC sub-state machines. If
the CP receives a further 'options'/’optionsResponse’ message, it MUST ignore the message and stay in the ACTIVE state.

Figure 10: CLUE Participant state machine
6.1. Media Provider’s state machine

As soon as the sub-state machine of the MP (Figure 11) is activated, it is in the ADV state. In the ADV state, the MP prepares the ‘advertisement’ message reflecting its actual telepresence capabilities.

After the ‘advertisement’ has been sent ("advertisement sent"), the MP moves from the ADV state to the WAIT FOR ACK state. If an ‘ack’ message with a successful response code arrives ("ack received"), the MP moves to the WAIT FOR CONF state. If a NACK arrives (i.e., an ‘ack’ message with an error response code), the MP moves back to the ADV state for preparing a new ‘advertisement’. When in the WAIT FOR ACK state, if a ‘configure’ message with the <ack> element set to TRUE arrives ("configure+ack received"), the MP goes directly to the CONF RESPONSE state. ‘configure+ack’ messages referring to out-of-date (i.e., having a sequence number less than the highest generated so far) advertisements MUST be ignored, i.e., they do not trigger any state transition. If the telepresence settings of the MP change while in the WAIT FOR ACK state ("changed telepresence settings"), the MP switches from the WAIT FOR ACK state to the ADV state to create a new ‘advertisement’.

When in the WAIT FOR CONF state, the MP listens to the channel for a ‘configure’ request coming from the MC. When a ‘configure’ arrives ("configure received"), the MP switches to the CONF RESPONSE state. If the telepresence settings change in the meanwhile ("changed telepresence settings"), the MP moves from the WAIT FOR CONF back to the ADV state to create the new ‘advertisement’ to be sent to the MC.

The MP in the CONF RESPONSE state processes the received ‘configure’ in order to produce a ‘configureResponse’ message. If the MP successfully processes the MC’s configuration, then it sends a 200 ‘configureResponse’ ("success configureResponse sent") and moves to the ESTABLISHED state. If there are errors in the ‘configure’ processing, then the MP issues a ‘configureResponse’ carrying an error response code and it goes back to the WAIT FOR CONF state to wait for a new configuration request. Finally, if there are changes in the MP’s telepresence settings ("changed telepresence settings"), the MP switches to the ADV state.

The MP in the ESTABLISHED state has successfully negotiated the media streams with the MC by means of the CLUE messages. If there are changes in the MP’s telepresence settings ("changed telepresence settings"), the MP moves back to the ADV state. In the ESTABLISHED state, the CLUE-controlled media streams of the session are those described in the last successfully processed ‘configure’ message.
Messages not shown for a state do not cause the state to change.

![State Machine Diagram]

**Figure 11: Media Provider’s state machine**

**6.2. Media Consumer’s state machine**

As soon as the sub-state machine of the MC (Figure 12) is activated, it is in the WAIT FOR ADV state. An MC in the WAIT FOR ADV state is waiting for an 'advertisement' coming from the MP. If the
‘advertisement’ arrives ("ADV received"), the MC reaches the ADV PROCESSING state. Otherwise, the MC stays in the WAIT FOR ADV state.

In the ADV PROCESSING state, the ‘advertisement’ is parsed by the MC. If the ‘advertisement’ is successfully processed, there are two possibilities. In the former case, the MC issues a successful ‘ack’ message to the MP ("ACK sent") and moves to the CONF state. This typically happens when the MC needs some more time to produce the ‘configure’ message associated with the received ‘advertisement’. In the latter case, the MC is able to immediately prepare and send back to the MP a ‘configure’ message. Such a message will have the <ack> field set to "200" ("configure+ack sent") and will allow the MC to move directly to the WAIT FOR CONF RESPONSE state.

If the ADV processing is unsuccessful (bad syntax, missing XML elements, etc.), the MC sends a NACK message (i.e., an ‘ack’ with an error response code) to the MP and optionally further describes the problem via a proper reason phrase. In this way ("NACK sent"), the MC switches back to the WAIT FOR ADV state, waiting for a new ‘advertisement’.

When in the CONF state, the MC prepares the ‘configure’ request to be issued to the MP on the basis of the previously ack-ed ‘advertisement’. When the ‘configure’ has been sent ("configure sent"), the MC moves to the WAIT FOR CONF RESPONSE state. If a new ‘advertisement’ arrives in the meanwhile ("advertisement received"), the MC goes back to the ADV PROCESSING state.

In the WAIT FOR CONF RESPONSE state, the MC waits for the MP’s response to the issued ‘configure’ or ‘configure+ack’. If a 200 ‘configureResponse’ message is received ("successful configureResponse received"), it means that the MP and the MC have successfully agreed on the media streams to be shared. Then, the MC can move to the ESTABLISHED state. On the other hand, if an error response is received ("error configureResponse received"), the MC moves back to the CONF state to prepare a new ‘configure’ request. If a new ‘advertisement’ is received in the WAIT FOR CONF RESPONSE state, the MC switches to the ADV PROCESSING state.

When the MC is in the ESTABLISHED state, the telepresence session configuration has been set up at the CLUE application level according to the MC’s preferences. Both the MP and the MC have agreed on (and are aware of) the CLUE-controlled media streams to be exchanged within the call. While in the ESTABLISHED state, it might happen that the MC decides to change something in the call settings. The MC then issues a new ‘configure’ ("configure sent") and goes to wait for the new ‘configureResponse’ in the WAIT FOR CONF RESPONSE state. On the other hand, in the ESTABLISHED state, if a new ‘advertisement’
arrives from the MP ("advertisement received"), it means that something has changed on the MP’s side. The MC then moves to the ADV PROCESSING state.

Messages not shown for a state do not cause the state to change.

Figure 12: Media Consumer’s state machine
7. Versioning

CLUE protocol messages are XML messages compliant to the CLUE protocol XML schema [I-D.ietf-clue-data-model-schema]. The version of the protocol corresponds to the version of the schema. Both client and server have to test the compliance of the received messages with the XML schema of the CLUE protocol. If the compliance is not verified, the message cannot be processed further.

Client and server cannot communicate if they do not share exactly the same XML schema. Such a schema is associated with the CLUE URN "urn:ietf:params:xml:ns:clue-protocol". If all CLUE-enabled devices use that schema there will be no interoperability problems due to schema issues.

This document defines XML schema version 1.0. The version usage is similar in philosophy to XMPP ([RFC6120]). A version number has major and minor components, each a non-negative integer. Major version changes denote non-interoperable changes. Minor version changes denote schema changes that are backward compatible by ignoring unknown XML elements, or other backward compatible changes.

The minor versions of the XML schema MUST be backward compatible, not only in terms of schema but also semantically and procedurally as well. This means that they should define further features and functionality besides those defined in the previous versions, in an incremental way, without impacting the basic rules defined in the previous version of the schema. In this way, if a MP is able to speak, e.g., version 1.5 of the protocol while the MC only understands version 1.4, the MP should have no problem in reverting the dialogue back to version 1.4 without exploiting 1.5 features and functionality. Version 1.4 is the one to be spoken and has to appear in the "v" attribute of the subsequent CLUE messages. In other words, in this example, the MP MUST use version 1.4. This said, and coherently with the general IETF "protocol robustness principle" stating that "an implementation must be conservative in its sending behavior, and liberal in its receiving behavior" ([RFC1122]), CLUE Participants MUST ignore unknown elements or attributes that are not envisioned in the negotiated protocol version and related extensions.

8. Extensions

Although the standard version of the CLUE protocol XML schema is designed to thoroughly cope with the requirements emerging from the application domain, new needs might arise and extensions can be designed. Extensions specify information and behaviors that are not described in a certain version of the protocol and specified in the related RFC document. Such information and behaviors can be
optionally used in a CLUE dialogue and MUST be negotiated in the CLUE initiation phase. They can relate to:

1. new information, to be carried in the existing messages. For example, more fields may be added within an existing message;

2. new messages. This is the case if there is no proper message for a certain task, so a brand new CLUE message needs to be defined.

As to the first type of extensions, it is possible to distinguish between protocol-specific and data model information. Indeed, CLUE messages are envelopes carrying both:

- (i) XML elements defined within the CLUE protocol XML schema itself (protocol-specific information)
- (ii) other XML elements compliant to the CLUE data model schema (data model information)

When new protocol-specific information is needed somewhere in the protocol messages, it can be added in place of the <any> elements and <anyAttribute> elements envisioned by the protocol schema. The policy currently defined in the protocol schema for handling <any> and <anyAttribute> elements is:

- elementFormDefault="qualified"
- attributeFormDefault="unqualified"

The new information must be qualified by namespaces other than "urn:ietf:params:xml:ns:clue-protocol" (the protocol URN) and "urn:ietf:params:xml:ns:clue-info" (the data model URN). Elements or attributes from unknown namespaces MUST be ignored.

The other matter concerns data model information. Data model information is defined by the XML schema associated with the URN "urn:ietf:params:xml:ns:clue-info". Also for the XML elements defined in such a schema there are extensibility issues. Those issues are overcome by using <any> and <anyAttribute> placeholders. New information within data model elements can be added in place of <any> and <anyAttribute> schema elements, as long as they are properly namespace qualified.

On the other hand (second type of extensions), "extra" CLUE protocol messages, i.e., messages not envisioned in the latest standard version of the schema, can be needed. In that case, the messages and the associated behavior should be defined in external documents that both communication parties must be aware of.
As reported in Figure 13, the fields of the <extension> element (either new information or new messages) take the following values:

- a name;
- an external XML Schema defining the XML information and/or the XML messages representing the extension;
- the major standard version of the protocol that the extension refers to.

```xml
<xs:complexType name="extensionType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="schemaRef" type="xs:anyURI"/>
    <xs:element name="version" type="versionType"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

Figure 13: The <extension> element

The above described <extension> element is carried within the ‘options’ and ‘optionsResponse’ messages to represent the extensions supported both by the CI and the CR.

Extensions MUST be defined in a separate XML schema file and MUST be provided with a companion document describing their semantics and use.

### 8.1. Extension example

An example of extension might be a "new" capture attribute (i.e., a capture attribute that is not envisioned in the current standard defining the CLUE data model in [I-D.ietf-clue-data-model-schema]) needed to further describe a video capture.

The CLUE data model document ([I-D.ietf-clue-data-model-schema]) envisions the possibility of adding this kind of "extra" information in the description of a video capture. For the sake of convenience, the XML definition of a video capture taken from [I-D.ietf-clue-data-model-schema] is reported in Figure 14 below.
<xs:complexType name="videoCaptureType">
  <xs:complexContent>
    <xs:extension base="tns:mediaCaptureType">
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

Figure 14: XML definition of a CLUE video capture

According to such a definition, a video capture might have, after the set of the generic media capture attributes, a set of new attributes defined elsewhere, i.e., in an XML schema defining an extension. The XML schema defining the extension might look like the following (Figure 15):
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema version="1.0"
  targetNamespace="http://example.extensions.com/myVideoExtensions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://example.extensions.com/myVideoExtensions"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <!--
  This is the new element to be put in place of the <any>
  element in the video capture definition
  of the CLUE data model schema
  -->
  <xs:element name="myVideoExtension">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="newVideoAttribute1"/>
        <xs:element ref="newVideoAttribute2"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="newVideoAttribute1" type="xs:string"/>
    <xs:element name = "newVideoAttribute2" type = "xs:boolean"/>
  </xs:element>
</xs:schema>

Figure 15: XML schema defining an extension

By using the extension above, a video capture can be further described in the advertisement using the <myVideoExtension> element containing two extra information (<newVideoAttribute1> and <newVideoAttribute2>) besides using the attributes envisioned for a generic media capture. As stated in this document, both participants must be aware of the extension schema and related semantics to use such an extension and must negotiate it via the 'options' and 'optionsResponse' mechanism.

9. XML Schema

NOTE TO THE RFC-Editor: Please replace "data-model-schema-19.xsd" with the right schema location for the CLUE data model schema document (which is still to be defined at the time of this writing) in this section prior to publication as an RFC.
In this section, the XML schema defining the CLUE messages is provided (Figure 16).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:ietf:params:xml:ns:clue-protocol"
    xmlns:dm="urn:ietf:params:xml:ns:clue-info"
    xmlns:tns="urn:ietf:params:xml:ns:clue-protocol"
    version="1.0"
    targetNamespace="urn:ietf:params:xml:ns:clue-protocol"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <!-- Import data model schema -->
    <xs:import namespace="urn:ietf:params:xml:ns:clue-info"
        schemaLocation="clue-data-model-schema-19.xsd" />
    <!-- ELEMENT DEFINITIONS -->
    <xs:element name="options" type="optionsMessageType" />
    <xs:element name="optionsResponse" type="optionsResponseMessageType"/>
    <xs:element name="advertisement" type="advertisementMessageType"/>
    <xs:element name="ack" type="advAcknowledgementMessageType"/>
    <xs:element name="configure" type="configureMessageType"/>
    <xs:element name="configureResponse" type="configureResponseMessageType"/>
    <!-- CLUE MESSAGE TYPE -->
    <xs:complexType name="clueMessageType" abstract="true">
        <xs:sequence>
            <xs:element name="clueId" type="xs:string" minOccurs="0" />
            <xs:element name="sequenceNr" type="xs:positiveInteger" />
        </xs:sequence>
        <xs:attribute name="protocol" type="xs:string" fixed="CLUE"
            use="required" />
        <xs:attribute name="v" type="versionType" use="required" />
    </xs:complexType>
    <!-- CLUE RESPONSE TYPE -->
    <xs:complexType name="clueResponseType">
        <xs:complexContent>
            <xs:extension base="clueMessageType">
                <xs:sequence>
                    <xs:element name="responseCode" type="responseCodeType" />
                    <xs:element name="reasonString" type="xs:string"
                        minOccurs="0" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- VERSION TYPE -->
    <xs:simpleType name="versionType">
        ...
    </xs:simpleType>
</xs:schema>
```
<xs:restriction base="xs:string">
  <xs:pattern value="[1-9][0-9]*\.[0-9]+" />
</xs:restriction>
</xs:simpleType>

<!-- RESPONSE CODE TYPE -->
<xs:simpleType name="responseCodeType">
  <xs:restriction base="xs:integer">
    <xs:pattern value="[1-9][0-9][0-9]" />
  </xs:restriction>
</xs:simpleType>

<!-- SUCCESS RESPONSE CODE TYPE -->
<xs:simpleType name="successResponseCodeType">
  <xs:restriction base="xs:integer">
    <xs:pattern value="2[0-9][0-9]" />
  </xs:restriction>
</xs:simpleType>

<!-- CLUE OPTIONS -->
<xs:complexType name="optionsMessageType">
  <xs:complexContent>
    <xs:extension base="clueMessageType">
      <xs:sequence>
        <xs:element name="mediaProvider" type="xs:boolean"/>
        <xs:element name="mediaConsumer" type="xs:boolean"/>
        <xs:element name="supportedVersions" type="versionsListType" minOccurs="0" />
        <xs:element name="supportedExtensions" type="extensionsListType" minOccurs="0" />
        <xs:any namespace="##other" processContents="lax" minOccurs="0" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- VERSIONS LIST TYPE -->
<xs:complexType name="versionsListType">
  <xs:sequence>
    <xs:element name="version" type="versionType" minOccurs="1" maxOccurs="unbounded"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<!-- EXTENSIONS LIST TYPE -->
<xs:complexType name="extensionsListType">
  <xs:sequence>
    <xs:element name="extension" type="extensionType" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<!-- EXTENSION TYPE -->
<xs:complexType name="extensionType">
  <xs:sequence>
    <xs:element name="name" type="xs:string" />
    <xs:element name="schemaRef" type="xs:anyURI" />
    <xs:element name="version" type="versionType" />
    <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax" />
</xs:complexType>
<!-- CLUE 'optionsResponse' -->
<xs:complexType name="optionsResponseMessageType">
  <xs:complexContent>
    <xs:extension base="clueResponseType">
      <xs:sequence>
        <xs:element name="mediaProvider" type="xs:boolean" minOccurs="0"/>
        <xs:element name="mediaConsumer" type="xs:boolean" minOccurs="0"/>
        <xs:element name="version" type="versionType" minOccurs="0"/>
        <xs:element name="commonExtensions" type="extensionsListType" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
<xs:complexType name="advertisementMessageType">
  <xs:complexContent>
    <xs:extension base="clueMessageType">
      <xs:sequence>
        <!-- mandatory -->
        <xs:element name="mediaCaptures" type="dm:mediaCapturesType"/>
        <xs:element name="encodingGroups" type="dm:encodingGroupsType"/>
        <xs:element name="captureScenes" type="dm:captureScenesType"/>
        <!-- optional -->
        <xs:element name="simultaneousSets" type="dm:simultaneousSetsType" minOccurs="0"/>
        <xs:element name="globalViews" type="dm:globalViewsType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="people" type="dm:peopleType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<!-- CLUE 'configure' MESSAGE TYPE -->
<xs:complexType name="configureMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="advSequenceNr" type="xs:positiveInteger" minOccurs="0"/>
<xs:element name="ack" type="successResponseCodeType" minOccurs="0"/>
<xs:element name="captureEncodings" type="dm:captureEncodingsType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:complexType>
<!-- 'configureResponse' MESSAGE TYPE -->
<xs:complexType name="configureResponseMessageType">
<xs:complexContent>
<xs:extension base="clueResponseType">
<xs:sequence>
<xs:element name="confSequenceNr" type="xs:positiveInteger" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:complexType>
<!-- 'ack' MESSAGE TYPE -->
<xs:complexType name="advAcknowledgementMessageType">
<xs:complexContent>
<xs:extension base="clueResponseType">
<xs:sequence>
<xs:element name="advSequenceNr" type="xs:positiveInteger" />
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:complexType>
10. Call flow example

In this section the CLUE protocol messages exchanged in the following call flow are detailed. Only one direction of media is shown for simplicity, as the other direction is precisely symmetric.
CP1

<table>
<thead>
<tr>
<th>1.options</th>
</tr>
</thead>
</table>

CP2

<table>
<thead>
<tr>
<th>2.optionsResponse</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>3.advertisement</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>4.configure+ack</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>5.confResponse</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>6.advertisement</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>7.ack</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>8.configure</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>9.confResponse</th>
</tr>
</thead>
</table>

...
Two CLUE Participants, CP1 and CP2, have successfully set up the CLUE channel according to document [I-D.ietf-clue-datachannel]. CP1 is the Channel Initiator (CI) and CP2 is the Channel Receiver (CR).

The initiation phase starts (negotiation of the CLUE protocol version and extensions). CP1, as the CI, sends to CP2 an ‘options’ message specifying the supported versions and extensions (Section 10.1). CP1 supports: (i) version 1.4 with extensions E1, E2 and E3, (ii) version 2.7 with extensions E4 and E5. Because of such capabilities, CP1 sends an ‘options’ message with the ‘v’ attribute set to 1.4 and specifies explicitly all the supported versions and extensions in the corresponding fields of the ‘options’ message. In the ‘options’ message, CP1 specifies also that it intends to act both as a MP and as a MC.

CP2 supports version 3.0, version 2.9 and version 1.9 of the CLUE protocol, each version without any extension. Version 2.7 is the best common choice. Given the received ‘options’ message, CP2 answers with an ‘optionsResponse’ message in which it specifies only version 2.7 as the agreed version of the CLUE protocol to be used, leaving blank the extensions part of the message to say that no extension will be used in the CLUE session (Section 10.2). In the ‘optionsResponse’ message, CP2 specifies also that it intends to act both as a MP and as a MC.

After the initiation phase is completed, CP1 and CP2 start their activity as MP and as MC. For the sake of simplicity, the following call flow focuses only on the dialogue between MP CP1 and MC CP2.

CP1 advertises a telepresence configuration like the one described in [I-D.ietf-clue-data-model-schema], Sec. Sample XML File, where there are (i) three main video streams captured by three cameras, the central one capable of capturing a zoomed-out view of the overall telepresence room, (ii) a multi-content capture of the loudest segment of the room, and (iii) one audio capture for the audio of the whole room (Section 10.3).

CP2 receives CP1’s ‘advertisement’ message and, after processing it, sends back to CP1 a ‘configure + ack’ message where it declares to be interested only in the multi-content capture and in the audio capture (Section 10.4).

CP1 answers to CP2’s ‘configure + ack’ message with a ‘configureResponse’ message including a response code ‘200 - Success’ to accept all CP2’s requests (Section 10.5).

To reflect the changes in its telepresence offer, CP1 issues a new ‘advertisement’ message to CP2 (Section 10.6), this time adding also
a composed capture made by a big picture representing the current speaker and two picture-in-picture boxes representing the previous speakers (see more details about the telepresence description in [I-D.ietf-clue-data-model-schema], Sec. MCC example).

CP2 acknowledges the second ‘advertisement’ message with an ‘ack’ message (Section 10.7).

In a second moment, CP2 changes the requested media streams from CP1 by sending to CP1 a ‘configure’ message replacing the previously selected video streams with the new composed media streams advertised by CP1 (Section 10.8). Media from the previous configuration continue to flow during the reconfiguration process.

Finally, CP1 accept the last requests of CP2 with a ‘confResponse’ message (Section 10.9)

We also remark that in the depicted flow three distinct sequence number spaces per sender are involved (two of which appear in the snippets since we only show one direction of media). The discontinuity between the sequence number values in Section 10.2 and Section 10.3 is hence correct.

10.1. CLUE message nr. 1: ‘options’
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<options xmlns="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns2="urn:ietf:params:xml:ns:clue-info"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="1.4">
    <clueId>CP1</clueId>
    <sequenceNr>51</sequenceNr>
    <mediaProvider>true</mediaProvider>
    <mediaConsumer>true</mediaConsumer>
    <supportedVersions>
        <version>1.4</version>
        <version>2.7</version>
    </supportedVersions>
    <supportedExtensions>
        <extension>
            <name>E1</name>
            <schemaRef>URL_E1</schemaRef>
            <version>1.4</version>
        </extension>
        <extension>
            <name>E2</name>
            <schemaRef>URL_E2</schemaRef>
            <version>1.4</version>
        </extension>
        <extension>
            <name>E3</name>
            <schemaRef>URL_E3</schemaRef>
            <version>1.4</version>
        </extension>
        <extension>
            <name>E4</name>
            <schemaRef>URL_E4</schemaRef>
            <version>2.7</version>
        </extension>
        <extension>
            <name>E5</name>
            <schemaRef>URL_E5</schemaRef>
            <version>2.7</version>
        </extension>
    </supportedExtensions>
</options>
10.2. CLUE message nr. 2: ‘optionsResponse’

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<optionsResponse xmlns="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns2="urn:ietf:params:xml:ns:clue-info"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:clue-protocol
http://wpage.unina.it/spromano/clue-protocol-17-schema-file.xsd"
protocol="CLUE" v="1.4">
  <clueId>CP2</clueId>
  <sequenceNr>62</sequenceNr>
  <responseCode>200</responseCode>
  <reasonString>Success</reasonString>
  <mediaProvider>true</mediaProvider>
  <mediaConsumer>true</mediaConsumer>
  <version>2.7</version>
</optionsResponse>
```

10.3. CLUE message nr. 3: ‘advertisement’

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:advertisement xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:clue-protocol
http://wpage.unina.it/spromano/clue-protocol-17-schema-file.xsd"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP1</ns2:clueId>
  <ns2:sequenceNr>11</ns2:sequenceNr>
  <ns2:mediaCaptures>
    <mediaCapture
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="audioCaptureType" captureID="AC0"
mediaType="audio">
      <captureSceneIDREF>CS1</captureSceneIDREF>
      <spatialInformation>
        <captureOrigin>
          <capturePoint>
            <x>0.0</x>
            <y>0.0</y>
            <z>10.0</z>
          </capturePoint>
          <lineOfCapturePoint>
```
<topRight>
<x>-1.0</x>
<y>20.0</y>
<z>11.0</z>
</topRight>
</captureArea>
</spatialInformation>
<individual>true</individual>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">left camera video capture</description>
<priority>1</priority>
<lang>it</lang>
<mobility>static</mobility>
&view>individual</view>
<capturedPeople>
<personIDREF>ciccio</personIDREF>
</capturedPeople>
</mediaCapture>
<mediaCapture
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="videoCaptureType" captureID="VC1" mediaType="video">
<captureSceneIDREF>CS1</captureSceneIDREF>
<spatialInformation>
<captureOrigin>
<capturePoint>
<x>0.0</x>
<y>0.0</y>
<z>10.0</z>
</capturePoint>
</captureOrigin>
<captureArea>
<bottomLeft>
<x>-1.0</x>
<y>20.0</y>
<z>9.0</z>
</bottomLeft>
<bottomRight>
<x>1.0</x>
<y>20.0</y>
<z>9.0</z>
</bottomRight>
<topLeft>
<x>-1.0</x>
<y>20.0</y>
<z>11.0</z>
</topLeft>
</captureArea>
<x>1.0</x>
<y>20.0</y>
<z>11.0</z>
</topRight>
</captureArea>
</spatialInformation>
<individual>true</individual>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">central camera video capture</description>
<priority>1</priority>
<lang>it</lang>
<mobility>static</mobility>
<view>individual</view>
<capturedPeople>
  <personIDREF>alice</personIDREF>
</capturedPeople>
</mediaCapture>
<description lang="en">right camera video capture</description>
<priority>1</priority>
<mobility>static</mobility>
<view>individual</view>
<capturedPeople>
  <personIDREF>bob</personIDREF>
</capturedPeople>
</mediaCapture>
<mediaCapture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="videoCaptureType" captureID="VC3" mediaType="video">
  <captureSceneIDREF>CS1</captureSceneIDREF>
  <spatialInformation>
    <captureArea>
      <bottomLeft>
        <x>-3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomLeft>
      <bottomRight>
        <x>3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomRight>
      <topLeft>
        <x>-3.0</x>
        <y>20.0</y>
        <z>11.0</z>
      </topLeft>
      <topRight>
        <x>3.0</x>
        <y>20.0</y>
        <z>11.0</z>
      </topRight>
    </captureArea>
    <content>
      <sceneViewIDREF>SE1</sceneViewIDREF>
    </content>
  </spatialInformation>
</mediaCapture>
<policy>SoundLevel:0</policy>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">loudest room segment</description>
<priority>2</priority>
<lang>it</lang>
<mobility>static</mobility>
<View>individual</View>
<mediaCapture>
  <xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="videoCaptureType" captureID="VC4" mediaType="video">
    <captureSceneIDREF>CS1</captureSceneIDREF>
    <spatialInformation>
      <captureOrigin>
        <capturePoint>
          <x>0.0</x>
          <y>0.0</y>
          <z>10.0</z>
        </capturePoint>
      </captureOrigin>
      <captureArea>
        <bottomLeft>
          <x>-3.0</x>
          <y>20.0</y>
          <z>7.0</z>
        </bottomLeft>
        <bottomRight>
          <x>3.0</x>
          <y>20.0</y>
          <z>7.0</z>
        </bottomRight>
        <topLeft>
          <x>-3.0</x>
          <y>20.0</y>
          <z>13.0</z>
        </topLeft>
        <topRight>
          <x>3.0</x>
          <y>20.0</y>
          <z>13.0</z>
        </topRight>
      </captureArea>
    </spatialInformation>
    <individual>true</individual>
    <encGroupIDREF>EG0</encGroupIDREF>
    <description lang="en">zoomed out view of all people in the room</description>
    <priority>2</priority>
<lang>it</lang>
<mobility>static</mobility>
&view>room</view>
<capturedPeople>
  <personIDREF>alice</personIDREF>
  <personIDREF>bob</personIDREF>
  <personIDREF>ciccio</personIDREF>
</capturedPeople>
</mediaCapture>
</ns2:mediaCaptures>
<ns2:encodingGroups>
  <encodingGroup encodingGroupID="EG0">
    <maxGroupBandwidth>600000</maxGroupBandwidth>
    <encodingIDList>
      <encodingID>ENC1</encodingID>
      <encodingID>ENC2</encodingID>
      <encodingID>ENC3</encodingID>
    </encodingIDList>
  </encodingGroup>
  <encodingGroup encodingGroupID="EG1">
    <maxGroupBandwidth>300000</maxGroupBandwidth>
    <encodingIDList>
      <encodingID>ENC4</encodingID>
      <encodingID>ENC5</encodingID>
    </encodingIDList>
  </encodingGroup>
</ns2:encodingGroups>
<ns2:captureScenes>
  <captureScene scale="unknown" sceneID="CS1">
    <sceneViews>
      <sceneView sceneViewID="SE1">
        <mediaCaptureIDs>
          <mediaCaptureIDREF>VC0</mediaCaptureIDREF>
          <mediaCaptureIDREF>VC1</mediaCaptureIDREF>
          <mediaCaptureIDREF>VC2</mediaCaptureIDREF>
        </mediaCaptureIDs>
      </sceneView>
      <sceneView sceneViewID="SE2">
        <mediaCaptureIDs>
          <mediaCaptureIDREF>VC3</mediaCaptureIDREF>
        </mediaCaptureIDs>
      </sceneView>
      <sceneView sceneViewID="SE3">
        <mediaCaptureIDs>
          <mediaCaptureIDREF>VC4</mediaCaptureIDREF>
        </mediaCaptureIDs>
      </sceneView>
      <sceneView sceneViewID="SE4">
      </sceneView>
    </sceneViews>
  </captureScene>
</ns2:captureScenes>
<mediaCaptureIDs>
  <mediaCaptureIDREF>AC0</mediaCaptureIDREF>
</mediaCaptureIDs>
</sceneView>
</captureScene>
</ns2:captureScenes>
<ns2:simultaneousSets>
  <simultaneousSet setID="SS1">
    <mediaCaptureIDREF>VC3</mediaCaptureIDREF>
    <sceneViewIDREF>SE1</sceneViewIDREF>
  </simultaneousSet>
  <simultaneousSet setID="SS2">
    <mediaCaptureIDREF>VC0</mediaCaptureIDREF>
    <mediaCaptureIDREF>VC2</mediaCaptureIDREF>
    <mediaCaptureIDREF>VC4</mediaCaptureIDREF>
  </simultaneousSet>
</ns2:simultaneousSets>
<ns2:people>
  <person personID="bob">
    <personInfo>
      <ns3:fn>
        Bob
      </ns3:fn>
    </personInfo>
    <personType>minute taker</personType>
  </person>
  <person personID="alice">
    <personInfo>
      <ns3:fn>
        Alice
      </ns3:fn>
    </personInfo>
    <personType>presenter</personType>
  </person>
  <person personID="ciccio">
    <personInfo>
      <ns3:fn>
        Ciccio
      </ns3:fn>
    </personInfo>
    <personType>chairman</personType>
    <personType>timekeeper</personType>
  </person>
</ns2:people>
</ns2:advertisement>
10.4. CLUE message nr. 4: ‘configure + ack’

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:configure xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:clue-protocol
http://wpage.unina.it/spromano/clue-protocol-17-schema-file.xsd"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP2</ns2:clueId>
  <ns2:sequenceNr>22</ns2:sequenceNr>
  <ns2:advSequenceNr>11</ns2:advSequenceNr>
  <ns2:ack>200</ns2:ack>
  <ns2:captureEncodings>
    <captureEncoding ID="ce123">
      <captureID>AC0</captureID>
      <encodingID>ENC4</encodingID>
    </captureEncoding>
    <captureEncoding ID="ce223">
      <captureID>VC3</captureID>
      <encodingID>ENC1</encodingID>
      <configuredContent>
        <sceneViewIDREF>SE1</sceneViewIDREF>
      </configuredContent>
    </captureEncoding>
  </ns2:captureEncodings>
</ns2:configure>

10.5. CLUE message nr. 5: ‘confResponse’
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:configureResponse xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP1</ns2:clueId>
  <ns2:sequenceNr>12</ns2:sequenceNr>
  <ns2:reasonString>Success</ns2:reasonString>
  <ns2:confSequenceNr>22</ns2:confSequenceNr>
</ns2:configureResponse>

10.6. CLUE message nr. 6: ‘advertisement’

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:advertisement xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP1</ns2:clueId>
  <ns2:sequenceNr>13</ns2:sequenceNr>
  <ns2:mediaCaptures>
    <mediaCapture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="audioCaptureType" captureID="AC0" mediaType="audio">
      <captureSceneIDREF>CS1</captureSceneIDREF>
      <spatialInformation>
        <captureOrigin>
          <capturePoint>
            <x>0.0</x>
            <y>0.0</y>
            <z>10.0</z>
          </capturePoint>
          <lineOfCapturePoint>
            <x>0.0</x>
            <y>1.0</y>
            <z>10.0</z>
          </lineOfCapturePoint>
        </captureOrigin>
      </spatialInformation>
    </mediaCapture>
  </ns2:mediaCaptures>
</ns2:advertisement>
<individual>true</individual>
<encGroupIDREF>EG1</encGroupIDREF>
<description lang="en">main audio from the room</description>
<priority>1</priority>
<lang>it</lang>
<mobility>static</mobility>
<view>room</view>
<capturedPeople>
  <personIDREF>alice</personIDREF>
  <personIDREF>bob</personIDREF>
  <personIDREF>ciccio</personIDREF>
</capturedPeople>
</mediaCapture>
<mediaCapture
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="videoCaptureType" captureID="VC0" mediaType="video">
  <captureSceneIDREF>CS1</captureSceneIDREF>
  <spatialInformation>
    <captureOrigin>
      <capturePoint>
        <x>0.5</x>
        <y>1.0</y>
        <z>0.5</z>
      </capturePoint>
      <lineOfCapturePoint>
        <x>0.5</x>
        <y>0.0</y>
        <z>0.5</z>
      </lineOfCapturePoint>
    </captureOrigin>
  </spatialInformation>
  <individual>true</individual>
  <encGroupIDREF>EG0</encGroupIDREF>
  <description lang="en">left camera video capture</description>
  <priority>1</priority>
  <lang>it</lang>
  <mobility>static</mobility>
  <view>individual</view>
  <capturedPeople>
    <personIDREF>ciccio</personIDREF>
  </capturedPeople>
</mediaCapture>
<mediaCapture
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:type="videoCaptureType" captureID="VC1" mediaType="video">
  <captureSceneIDREF>CS1</captureSceneIDREF>
  <spatialInformation>
    <captureOrigin>
      <capturePoint>
        <x>0.5</x>
        <y>1.0</y>
        <z>0.5</z>
      </capturePoint>
      <lineOfCapturePoint>
        <x>0.5</x>
        <y>0.0</y>
        <z>0.5</z>
      </lineOfCapturePoint>
    </captureOrigin>
  </spatialInformation>
  <individual>true</individual>
  <encGroupIDREF>EG0</encGroupIDREF>
  <description lang="en">left camera video capture</description>
  <priority>1</priority>
  <lang>it</lang>
  <mobility>static</mobility>
  <view>individual</view>
  <capturedPeople>
    <personIDREF>ciccio</personIDREF>
  </capturedPeople>
</mediaCapture>
<spatialInformation>
  <captureOrigin>
    <capturePoint>
      <x>0.0</x>
      <y>0.0</y>
      <z>10.0</z>
    </capturePoint>
  </captureOrigin>
  <captureArea>
    <bottomLeft>
      <x>-1.0</x>
      <y>20.0</y>
      <z>9.0</z>
    </bottomLeft>
    <bottomRight>
      <x>1.0</x>
      <y>20.0</y>
      <z>9.0</z>
    </bottomRight>
    <topLeft>
      <x>-1.0</x>
      <y>20.0</y>
      <z>11.0</z>
    </topLeft>
    <topRight>
      <x>1.0</x>
      <y>20.0</y>
      <z>11.0</z>
    </topRight>
  </captureArea>
</spatialInformation>
<individual>true</individual>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">central camera video capture</description>
<priority>1</priority>
<lang>it</lang>
<mobility>static</mobility>
$view>individual</view>
<capturedPeople>
  <personIDREF>alice</personIDREF>
</capturedPeople>
</mediaCapture>
<captureOrigin>
  <capturePoint>
    <x>2.0</x>
    <y>0.0</y>
    <z>10.0</z>
  </capturePoint>
</captureOrigin>
<captureArea>
  <bottomLeft>
    <x>1.0</x>
    <y>20.0</y>
    <z>9.0</z>
  </bottomLeft>
  <bottomRight>
    <x>3.0</x>
    <y>20.0</y>
    <z>9.0</z>
  </bottomRight>
  <topLeft>
    <x>1.0</x>
    <y>20.0</y>
    <z>11.0</z>
  </topLeft>
  <topRight>
    <x>3.0</x>
    <y>20.0</y>
    <z>11.0</z>
  </topRight>
</captureArea>
</spatialInformation>
<individual>true</individual>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">right camera video capture</description>
<priority>1</priority>
<lang>it</lang>
<mobility>static</mobility>
&view>individual</view>
<capturedPeople>
  <personIDREF>bob</personIDREF>
</capturedPeople>
</mediaCapture>
<bottomLeft>
  <x>-3.0</x>
  <y>20.0</y>
  <z>9.0</z>
</bottomLeft>
<bottomRight>
  <x>3.0</x>
  <y>20.0</y>
  <z>9.0</z>
</bottomRight>
<topLeft>
  <x>-3.0</x>
  <y>20.0</y>
  <z>11.0</z>
</topLeft>
<topRight>
  <x>3.0</x>
  <y>20.0</y>
  <z>11.0</z>
</topRight>
</captureArea>
</spatialInformation>
<content>
  <sceneViewIDREF>SE1</sceneViewIDREF>
</content>
<policy>SoundLevel:0</policy>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">loudest room segment</description>
<priority>2</priority>
/lang>it</lang>
<mobility>static</mobility>
<view>individual</view>
</mediaCapture>
<mediaCapture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="videoCaptureType" captureID="VC5" mediaType="video">
  <captureSceneIDREF>CS1</captureSceneIDREF>
  <spatialInformation>
    <captureArea>
      <bottomLeft>
        <x>-3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomLeft>
      <bottomRight>
        <x>3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomRight>
    </captureArea>
  </spatialInformation>
</mediaCapture>

<mediaCapture xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="videoCaptureType" captureID="VC1" mediaType="video">
  <captureSceneIDREF>CS1</captureSceneIDREF>
  <spatialInformation>
    <captureArea>
      <bottomLeft>
        <x>-3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomLeft>
      <bottomRight>
        <x>3.0</x>
        <y>20.0</y>
        <z>9.0</z>
      </bottomRight>
    </captureArea>
  </spatialInformation>
</mediaCapture>

<individual>true</individual>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">zoomed out view of all people in the room</description>
<priority>2</priority>
<lang>it</lang>
<mobility>static</mobility>
</mediaCapture>
<y>20.0</y>
<z>9.0</z>
</bottomRight>
<topLeft>
<x>-3.0</x>
<y>20.0</y>
<z>11.0</z>
</topLeft>
<topRight>
<x>3.0</x>
<y>20.0</y>
<z>11.0</z>
</topRight>

</captureArea>
</spatialInformation>
<content>
<sceneViewIDREF>SE1</sceneViewIDREF>
</content>
<policy>SoundLevel:1</policy>
<description lang="en">penultimate loudest room segment</description>
<lang>it</lang>
<mobility>static</mobility>
</mediaCapture>
<mediaCapture
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="videoCaptureType" captureID="VC6" mediaType="video">
<captureSceneIDREF>CS1</captureSceneIDREF>
</mediaCapture>
<spatialInformation>
<captureArea>
<bottomLeft>
<x>-3.0</x>
<y>20.0</y>
<z>9.0</z>
</bottomLeft>
</captureArea>
</mediaCapture>
<y>20.0</y>
<z>11.0</z>
</topRight>
</spatialInformation>
<content>
<sceneViewIDREF>SE1</sceneViewIDREF>
</content>
<policy>SoundLevel:2</policy>
<description lang="en">last but two loudest room segment</description>
<lang>it</lang>
<mobility>static</mobility>
&view>individual</view>
</mediaCapture>
<mediaCapture
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="videoCaptureType" captureID="VC7" mediaType="video">
<captureSceneIDREF>CS1</captureSceneIDREF>
<spatialInformation>
<captureArea>
<bottomLeft>
<x>-3.0</x>
<y>20.0</y>
<z>9.0</z>
</bottomLeft>
<bottomRight>
<x>3.0</x>
<y>20.0</y>
<z>9.0</z>
</bottomRight>
<topLeft>
<x>-3.0</x>
<y>20.0</y>
<z>11.0</z>
</topLeft>
<topRight>
<x>3.0</x>
<y>20.0</y>
<z>11.0</z>
</topRight>
</captureArea>
</spatialInformation>
<content>
<mediaCaptureIDREF>VC3</mediaCaptureIDREF>
<mediaCaptureIDREF>VC5</mediaCaptureIDREF>
<mediaCaptureIDREF>VC6</mediaCaptureIDREF>
</content>
<maxCaptures exactNumber="true">3</maxCaptures>
<encGroupIDREF>EG0</encGroupIDREF>
<description lang="en">big picture of the current speaker +
pips about previous speakers</description>
<priority>3</priority>
<lang>it</lang>
<mobility>static</mobility>
&view>individual</view>
</mediaCapture>
</ns2:mediaCaptures>
<ns2:encodingGroups>
<encodingGroup encodingGroupID="EG0">
<maxGroupBandwidth>600000</maxGroupBandwidth>
<encodingIDList>
  <encodingID>ENC1</encodingID>
  <encodingID>ENC2</encodingID>
  <encodingID>ENC3</encodingID>
</encodingIDList>
</encodingGroup>
<encodingGroup encodingGroupID="EG1">
<maxGroupBandwidth>300000</maxGroupBandwidth>
<encodingIDList>
  <encodingID>ENC4</encodingID>
  <encodingID>ENC5</encodingID>
</encodingIDList>
</encodingGroup>
</ns2:encodingGroups>
<ns2:captureScenes>
<captureScene scale="unknown" sceneID="CS1">
  <sceneViews>
    <sceneView sceneViewID="SE1">
      <description lang="en">participants’ individual
videos</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC0</mediaCaptureIDREF>
        <mediaCaptureIDREF>VC1</mediaCaptureIDREF>
        <mediaCaptureIDREF>VC2</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
    <sceneView sceneViewID="SE2">
      <description lang="en">loudest segment of the
room</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC3</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
    <sceneView sceneViewID="SE5">
      <description lang="en">loudest segment of the
room</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC6</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
  </sceneViews>
</captureScene>
</ns2:captureScenes>
<ns2:captureScenes>
  <ns2:sceneViews>
    <sceneView sceneViewID="SE1">
      <description lang="en">room + pips</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC7</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
  </ns2:sceneViews>
  <ns2:sceneViews>
    <sceneView sceneViewID="SE2">
      <description lang="en">room audio</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC0</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
  </ns2:sceneViews>
  <ns2:sceneViews>
    <sceneView sceneViewID="SE3">
      <description lang="en">room video</description>
      <mediaCaptureIDs>
        <mediaCaptureIDREF>VC4</mediaCaptureIDREF>
      </mediaCaptureIDs>
    </sceneView>
  </ns2:sceneViews>
  <ns2:simultaneousSets>
    <simultaneousSet setID="SS1">
      <mediaCaptureIDREF>VC3</mediaCaptureIDREF>
      <mediaCaptureIDREF>VC7</mediaCaptureIDREF>
      <sceneViewIDREF>SE1</sceneViewIDREF>
    </simultaneousSet>
    <simultaneousSet setID="SS2">
      <mediaCaptureIDREF>VC0</mediaCaptureIDREF>
      <mediaCaptureIDREF>VC2</mediaCaptureIDREF>
      <mediaCaptureIDREF>VC4</mediaCaptureIDREF>
    </simultaneousSet>
  </ns2:simultaneousSets>
  <ns2:people>
    <person personID="bob">
      <personInfo>
        <ns3:fn>
          <ns3:text>Bob</ns3:text>
        </ns3:fn>
      </personInfo>
      <personType>minute taker</personType>
    </person>
    <person personID="alice">
      <personInfo>
        <ns3:fn>
          <ns3:text>Alice</ns3:text>
        </ns3:fn>
      </personInfo>
      <personType>presenter</personType>
    </person>
  </ns2:people>
</ns2:captureScenes>
10.7. CLUE message nr. 7: ‘ack’

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ack xmlns="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns2="urn:ietf:params:xml:ns:clue-info"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="2.7">
  <clueId>CP2</clueId>
  <sequenceNr>23</sequenceNr>
  <responseCode>200</responseCode>
  <advSequenceNr>13</advSequenceNr>
  <reasonString>Success</reasonString>
</ack>
```

10.8. CLUE message nr. 8: ‘configure’
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:configure xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP2</ns2:clueId>
  <ns2:sequenceNr>24</ns2:sequenceNr>
  <ns2:advSequenceNr>13</ns2:advSequenceNr>
  <ns2:captureEncodings>
    <captureEncoding ID="ce123">
      <captureID>AC0</captureID>
      <encodingID>ENC4</encodingID>
    </captureEncoding>
    <captureEncoding ID="ce456">
      <captureID>VC7</captureID>
      <encodingID>ENC1</encodingID>
      <configuredContent>
        <sceneViewIDREF>SE5</sceneViewIDREF>
      </configuredContent>
    </captureEncoding>
  </ns2:captureEncodings>
</ns2:configure>

10.9. CLUE message nr. 9: ‘confResponse’

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:configureResponse xmlns="urn:ietf:params:xml:ns:clue-info"
xmlns:ns2="urn:ietf:params:xml:ns:clue-protocol"
xmlns:ns3="urn:ietf:params:xml:ns:vcard-4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
protocol="CLUE" v="2.7">
  <ns2:clueId>CP1</ns2:clueId>
  <ns2:sequenceNr>14</ns2:sequenceNr>
  <ns2:confSequenceNr>24</ns2:confSequenceNr>
</ns2:configureResponse>
11. Security Considerations

As a general consideration, we remark that the CLUE framework (and related protocol) has been conceived at the outset by embracing the security-by-design paradigm. This entails that a number of requirements have been identified and properly standardized as mandatory within the entire set of documents associated with the CLUE architecture. Requirements include: (i) the use of cryptography and authentication; (ii) protection of all sensitive fields; (iii) mutual authentication between CLUE endpoints; (iv) the presence of authorization mechanisms; (v) the presence of native defence mechanisms against malicious activities such as eavesdropping, selective modification, deletion, replay (and related combinations thereof). Hence, security of the single components of the CLUE solution cannot be evaluated independently of the integrated view of the final architecture.

The CLUE protocol is an application-level protocol allowing a Media Producer and a Media Consumer to negotiate a variegated set of parameters associated with the establishment of a telepresence session. This unavoidably exposes a CLUE-enabled telepresence system to a number of potential threats, most of which are extensively discussed in the framework document [I-D.ietf-clue-framework]. The security considerations section of the mentioned document actually discusses issues associated with the setup and management of a telepresence session both in the basic case involving two CLUE endpoints acting, respectively, as MP and MC, and in the more advanced scenario envisaging the presence of an MCU.

The framework document also mentions that the information carried within CLUE protocol messages might contain sensitive data, which SHOULD hence be accessed only by authenticated endpoints. Security issues associated with the CLUE data model schema are discussed in [I-D.ietf-clue-data-model-schema].

There is extra information carried by the CLUE protocol that is not associated with the CLUE data model schema and which exposes information that might be of concern. This information is primarily exchanged during the negotiation phase via the ‘options’ and ‘optionsResponse’ messages. In the CLUE Participant state machine OPTIONS state, both parties agree on the version and on the extensions to be used in the subsequent CLUE messages exchange phase. A malicious participant might either try to retrieve a detailed footprint of a specific CLUE protocol implementation during this initial setup phase, or force the communicating party to use a non-up-to-date version of the protocol which they know how to break. Indeed, exposing all of the supported versions and extensions could conceivably leak information about the specific implementation of the
protocol. In theory an implementation could choose not to announce all of the versions it supports if it wants to avoid such leakage, though at the expenses of interoperability. With respect to the above considerations, it is noted that the OPTIONS state is only reached after the CLUE data channel has been successfully set up. This ensures that only authenticated parties can exchange ‘options’ and related ‘optionsResponse’ messages and hence drastically reduces the attack surface that is exposed to malicious parties.

The CLUE framework clearly states the requirement to protect CLUE protocol messages against threats deriving from the presence of a malicious agent capable to gain access to the CLUE data channel. Such a requirement is met by the CLUE data channel solution described in [I-D.ietf-clue-datachannel], which ensures protection from both message recovery and message tampering. With respect to this last point, any implementation of the CLUE protocol compliant with the CLUE specification MUST rely on the exchange of messages that flow on top of a reliable and ordered SCTP over DTLS transport channel connecting two CLUE Participants.

12. IANA Considerations

This document registers a new XML namespace, a new XML schema and the MIME type for the schema. This document also registers the "CLUE" Application Service tag and the "CLUE" Application Protocol tag and defines registries for the CLUE messages and response codes.

12.1. URN Sub-Namespace Registration

This section registers a new XML namespace, "urn:ietf:params:xml:ns:clue-protocol".


Registrant Contact: IESG (iesg@ietf.org).

XML:
12.2. XML Schema registration

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


Registrant Contact: IESG (iesg@ietf.org).

Schema: The XML for this schema can be found as the entirety of Section 9 of this document.

12.3. MIME Media Type Registration for ‘application/clue+xml’

This section registers the "application/clue+xml" MIME type.

To: ietf-types@iana.org

Subject: Registration of MIME media type application/clue+xml

MIME media type name: application

MIME subtype name: clue+xml

Required parameters: (none)

Optional parameters: charset
Same as the charset parameter of "application/xml" as specified in [RFC7303], Section 3.2.
Encoding considerations: Same as the encoding considerations of "application/xml" as specified in [RFC7303], Section 3.2.

Security considerations: This content type is designed to carry protocol data related to telepresence session control. Some of the data could be considered private. This media type does not provide any protection and thus other mechanisms such as those described in Section Security are required to protect the data. This media type does not contain executable content.

Interoperability considerations: None.

Published specification: RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]]

Applications that use this media type: CLUE participants.

Additional Information: Magic Number(s): (none), File extension(s): .xml, Macintosh File Type Code(s): TEXT.

Person & email address to contact for further information: Simon Pietro Romano (spromano@unina.it).

Intended usage: LIMITED USE

Author/Change controller: The IETF

Other information: This media type is a specialization of application/xml [RFC7303], and many of the considerations described there also apply to application/clue+xml.

12.4. CLUE Protocol Registry

The document requests that the IANA creates new registries for CLUE messages and response codes.

12.4.1. CLUE Message Types

The following summarizes the registry for CLUE messages:

Related Registry: CLUE Message Types Registry

Defining RFC: RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]]
Registration/Assignment Procedures: Following the policies outlined in [RFC8126], the IANA policy for assigning new values for the CLUE message types for the CLUE protocol is Specification Required.

Registrant Contact: IESG (iesg@ietf.org).

The initial Message table is populated using the CLUE messages described in Section 5 and defined in the XML schema in Section 9.

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Sent by the CI to the CR in the initiation phase to specify the roles played by the CI, the supported versions and the supported extensions.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>optionsResponse</td>
<td>Sent by the CI to the CR in reply to an ‘options’ message to finally establish the version and the extensions to be used in the following CLUE messages exchange.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>advertisement</td>
<td>Sent by the MP to the MC to specify the telepresence capabilities of the MP expressed according to the CLUE framework.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>ack</td>
<td>Sent by the MC to the MP to acknowledge the reception of an ‘advertisement’ message.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>configure</td>
<td>Sent by the MC to the MP to specify the desired media captures among those specified in the ‘advertisement’.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>configureResponse</td>
<td>Sent by the MP to the MC in reply to a CONFIGURE message to communicate if the configuration request has been successfully processed or not.</td>
<td>RFCXXXX</td>
</tr>
</tbody>
</table>

IANA-CLUE

12.4.2.  CLUE Response Codes

The following summarizes the requested registry for CLUE response codes:

Related Registry: CLUE Response Code Registry
Defining RFC: RFC XXXX [[NOTE TO IANA/RFC-EDITOR: Please replace XXXX with the RFC number for this specification.]]

Registration/Assignment Procedures: Following the policies outlined in [RFC8126], the IANA policy for assigning new values for the Response codes for CLUE shall be Specification Required.

Registrant Contact: IESG (iesg@ietf.org).

The initial Response-code table is populated using the Response codes defined in Section 5.7 as follows:

<table>
<thead>
<tr>
<th>Number</th>
<th>Default Response String</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Success</td>
<td>The request has been successfully processed.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>300</td>
<td>Low-level request error.</td>
<td>A generic low-level request error has occurred.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>301</td>
<td>Bad syntax</td>
<td>The XML syntax of the message is not correct.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>302</td>
<td>Invalid value</td>
<td>The message contains an invalid parameter value.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>303</td>
<td>Conflicting values</td>
<td>The message contains values that cannot be used together.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>400</td>
<td>Semantic errors</td>
<td>Semantic errors in the received CLUE protocol message.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>401</td>
<td>Version not supported</td>
<td>The protocol version used in the message is not supported.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>402</td>
<td>Invalid sequencing</td>
<td>Sequence number gap; repeated sequence number; sequence number outdated.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>403</td>
<td>Invalid identifier</td>
<td>The clueId used in the message is not valid or unknown.</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>404</td>
<td>The sequence number of the advertisement the configure refers to is out of date.</td>
<td>RFCXXXX</td>
<td></td>
</tr>
<tr>
<td>405</td>
<td>Subset choice not allowed</td>
<td>The subset choice is not allowed for the specified Multiple Content Capture.</td>
<td>RFCXXXX</td>
</tr>
</tbody>
</table>
13. Acknowledgments

The authors thank all the CLUErs for their precious feedbacks and support, in particular Paul Kyzivat, Christian Groves and Scarlett Liuyan.

14. References

14.1. Normative References

[I-D.ietf-clue-data-model-schema]

[I-D.ietf-clue-datachannel]

[I-D.ietf-clue-framework]

[I-D.ietf-clue-signaling]


14.2. Informative References


Authors' Addresses