COI Client Specification v1.0

Note: the UPDATE client specification is now available here:

https://github.com/coi-dev/coi-specs/blob/master/coi-client-spec.md

-- THE FOLLOWING SPECIFICATION IS OUTDATED AND ARCHIVED FOR NOW JUST FOR COMPARISON PURPOSES --

Status: DRAFT

Introduction

COI - Chat Over IMAP - provides a basis for realizing modern messaging services on top of the existing email infrastructure. COI leverages the existing open and federated email infrastructure to provide an open chat functionality accessible to everyone with an email address.

COI will work with standard IMAP servers and IMAP clients and will work better with COI-aware clients and servers. Initially, most email servers will not provide support for COI, so we need ways how to realize the most important functions on top of the existing infrastructure.

Conventions Used in This Document

In examples, "C:" indicates lines sent by a client that is connected to a server. "S:" indicates lines sent by the server to the client.

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.

COI Actors

In a typical COI setup the following actors are involved:

- A COI-compatible client, i.e. your app. Also called just "COI client".
- An service for retrieving the message data, currently we use IMAP for that, but COI is not tied to IMAP and could be made available through JMAP too, for example. The IMAP server may or may not be COI-compatible. The IMAP service MUST be compatible with IMAP v4rev1 as defined in RFC 3501, which is the case for almost any IMAP service in production.
- An SMTP service for sending messages. Sending services are also called Mail Transfer Agents (MTAs).
- Other clients, which may or may not be COI-compatible. Clients are also called Mail User Agents (MUAs).

Ideally, your client will be able to communicate with both COI-compatible and normal email clients, however this decision is your choice. You can also choose to only process COI-compliant messages in your app, which will reduce efforts but it might also reduce the potential audience for your users.

A typical COI communication sequence includes the following steps:

- Connect to your IMAP service and check for COI compliance - if it is supported, enable it for the currently signed in user.
- Send specifically formatted mail messages over SMTP.
- Listen for incoming messages, differentiate between chat, normal and blocked messages and move them to the respective folders.
- Show chat messages to the end user.

Checking for Server Side COI Support

A COI client should check for COI support from the server first. it will CAPABILITY after having logged into the IMAP service. If the returned capabilities list contains the "COI" capability, then the client does not need to filter messages on the client side. It will also have more advanced options at its disposal, see the server side specifications for details.

After signing into the IMAP service with your user credentials, call CAPABILITY. When the server returns the “COI” capability you will know that this is supported. You will find out configuration values and activate COI support for your user by calling ENABLE COI.

Note that you can “tag” your IMAP commands to easier differentiate responses from the IMAP service for commands that may run for a longer time. In this simplified example we use “a”, “b”, “c” and “d” as tags. Lines starting with "S:" are coming from the server, lines with "C:" come from the client:

```
S: * OK IMAP Server
C: a CAPABILITY
S: * CAPABILITY IMAP4rev1
S: a OK
C: b LOGIN foo bar
S: b OK Authentication successful.
C: c CAPABILITY
S: * CAPABILITY IMAP4rev1 COI ENABLE
S: c OK
C: d ENABLE COI
S: * COI MAILBOX-ROOT "COI"
S: * COI SUBMISSION "imap.example.com" "imap/alpn"
S: * COI SUBMISSION "mail.example.com" "smtp"
S: * COI VAPID-PUBLIC-KEY "abcdefg"
S: * COI VERSION 1
S: d OK COI enabled.
```

Message Format

Any chat message MUST be compliant with the RFC 5322 definition. If binary data is transmitted, the message additionally needs to conform to the MIME message standard (RFC 2045, RFC 2046, and RFC2049).

Message Format Design Considerations

Goals

With the COI specification we try to achieve the following goals:

- Backwards Compatibility
  Always have something to show for existing email clients. Each message SHOULD contain
  - at least one text/plain part,
  - additionally optionally a text/html part, or
  - a binary part like image/jpeg.
- Open Standards
  Use open and existing standards when possible, e.g.
  - VCard for contact information
  - Disposition Notification Reports for read receipts
  - application/json attachments for structured data
- Secure Messaging
  Support users to message securely,
  - Allow any encryption mechanism
  - Recommend autocrypt.org support
  - Optional direct IMAP 2 IMAP communication
- Modern User Experience.
  Provide support for allowing a user experience that most people expect, e.g.
  - Easy cross device contact sync
  - Allow to block users
  - Edit & delete sent messages
  - Useful interactions like polls
  - Push (with COI enabled backend)
  - Typing notifications (when both parties use a COI-enabled servers)

Backwards Compatibility

COI messages are designed to be backwards compatible with existing email clients. While some COI functionalities will not work on such legacy clients, users of non-COI-compliant clients should still be informed about it. Therefore, each message MUST have at least one part that explains the content of this message. This MUST be a text/plain part plus one of the following mine types:

- text/html, or
- a common binary format like for example image/jpeg.

For the same reason, the number of messages SHOULD be limited - ideally, instead of sending automated messages, a COI client only sends out user-generated messages and possibly enriches those.

If possible, commonly understood MIME parts such as image/jpeg or text/vcard are used, so that again non-COI compliant clients can at least view nested content parts.

COI-specific Message Header Fields

COI-aware email clients and COI IMAP servers MUST send messages conforming to the header descriptions below:

- A COI message MUST contain the "Chat-Version" header field with a value of "1.0". Note that further parameters MAY follow which will be separated by semicolons, e.g. "Chat-Version: 1.0;name=value".
- A COI message MUST only contain one address in the "From" field.
- A COI message MUST contain a "Date" header field with a value adhering to RFC 5322.
- A COI message MUST contain a "Message-ID" header field in the following format: "coi$" + domain-unique ID + "@" + mail domain, e.g. "coi$KJKJK2312321312KJ@mydomain.org".
- A COI message MUST contain "MIME-Version: 1.0" in the header.
- A COI message SHOULD contain a "References" header field with the message-IDs of the initial message and the last message only. This is the "References" header value of the parent message, if this is not set, the message-ID of the parent message. Note that this recommendations is contrary to the RFC 5322 recommendation, which results in all message-ID are all parent messages being referenced. However, in a chat conversation there will be way more messages than in a typical email thread, so it would be impractical to reference all previous messages.
- A COI message SHOULD contain a "In-Reply-To" header when answering to a previous message. This header field contains the message-ID in of the replied message.
- The "To" header contains the recipient(s) of the message.
- The "From" header field of a COI message MUST only contain one sender.
- A COI message SHOULD NOT have a "Sender" header field.
- A COI message MAY contain a "Disposition-Notification-To" header field. If it is present the field content MUST be the same as the FROM header field. This is for requesting read reports.
- A binary COI message MUST contain the "Content-Transfer-Encoding" header field.
- A binary COI message MUST contain the "Content-Disposition: inline" header. It SHOULD also contain the filename and size parameters, for example "Content-Disposition: inline;filename="earth.jpg";size=2048".
- The "Chat-Content" header MAY be used in message parts to identify the contents of a specific part.
- Depending on the message contents or its encryption, additional headers MAY be set.
- A COI client MAY set the "Subject" header field.

COI Base Formats

A COI message will use one of these formats as specified in the "Content-Type" header field:

- text/plain: for normal, user generated text messages without any additional data
- multipart/alternative: for user generated messages in which some parts may be applicable for a COI client while the text/plain or text/html part is applicable for both non-COI-compliant clients and COI clients.
- multipart/mixed: for creating more complex messages, e.g. the first message to a group that also contains a text/vcard part of the sender as an attachment.

Messages

Text Messages

- User generated text messages SHOULD be delivered in the TEXT/PLAIN mime format with a UTF-8 encoding when possible. This is defined with following header field: "Content-Type: text/plain; charset=UTF-8; format=flowed".
- Note that incoming messages from existing email clients MAY be delivered in other formats but SHOULD still be shown. Any existing signatures and quote blocks SHOULD be either stripped away or folded out of view. A client MAY show an option to view the full message contents.

Example:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8; format=flowed
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>

Hello World
My dear friend, this is a message to you!
```

NOTES: User case scenario: contains bold/colored/text use html only?

To allow clients to market themselves, user generated messages MAY be sent as multipart/alternative messages. In that case a COI client SHOULD prefer the text/plain part, whereas existing non-COI compliant clients will typically show the HTML content instead. NOT PART FOR THE SPEC? CREATE MULTIPART/MIXED with several parts

Example for a multipart/alternative message:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$22938.8238702@sample.com>
Content-Type: multipart/alternative;
 boundary=unique-boundary-1
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>
MIME-Version: 1.0
--unique-boundary-1
Content-Type: text/plain; Charset=UTF-8

hello COI world!

--unique-boundary-1
Content-Type: text/html; Charset=UTF-8

<p>hello COI world!</p>
--unique-boundary-1
Content-Type: text/html; charset=UTF-8
Chat-Content: ignore

<p><i>This message is a chat message - consider using <a href="https://myawesomecoiapp.com">my awesome COI app<
/a> for best experience!</i></p>

-unique-boundary-1--
```

An initial message SHOULD contain contact information about the sender, please compare the Contact Messages section below for details.

Binary Messages

Typical examples for binary messages are

- photos: image/webp, image/jpeg and image/png SHOULD be supported by a COI-compatible client. image/gif is RECOMMENDED to be supported.
- audio-recordings: audio/webm SHOULD be supported by a COI-compatible client. audio/mp4 and audio/aac is RECOMMENDED to be supported.
- videos: video/webm SHOULD be supported by a COI-compatible client. video/mp4 is RECOMMENDED to be supported.

In a chat environment, a single message typically only contains one binary message, but it MAY also contain several attachments at once.

A binary message MUST contain the following additional fields:

- Content-Transfer-Encoding: the encoding of the message, base64 MUST be supported, compare RFC 4648 for details.
- Content-Disposition: the disposition of the binary data. It SHOULD have the value "inline". Additionally the parameters "filename" and "size" (in octets=bytes) SHOULD be defined. Compare RFC 2183 for details.

A COI-compatible client SHOULD support messages with multiple files. In this case, the corresponding multipart/mixed messages need to be supported. If the COI client supports encryption, additionally multipart/encrypted messages MUST be supported.

Example 1: A message containing a single image.

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$434571BC.8070702@sample.com>
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
Content-Disposition: inline;filename=earth.jpg;size=2048
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>
MIME-Version: 1.0

TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=
```

Example 2: A multipart message containing one audio recording and one image.

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$123AB223.8238702@sample.com>
Content-Type: multipart/mixed;
              boundary=unique-boundary-1
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>
MIME-Version: 1.0

--unique-boundary-1
Content-Type: audio/webm
Content-Transfer-Encoding: base64
Content-Disposition: inline

... base64-encoded WebM audio data goes here ...

--unique-boundary-1
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
Content-Disposition: inline;
 filename="venus.jpg";size=4096

... base64-encoded JPEG image data goes here ...

--unique-boundary-1--
```

Preview Message Parts

For a large binary message such as a video or a high resolution photo, a COI-compatible client MAY add a preview message part. This allows clients with unstable or rate-limited network connections to show the message without needing to download the full message contents first.

A COI-compatible client MUST be able to process such a preview part and not show two individual parts. It MAY ignore such message parts to be compliant.

A preview message sets the "Chat-Content" header field to "preview" in the respective part:

```
 Chat-Content: preview
```

A preview message part typically contains one lower resolution image.

Edit and Delete Messages

To allow users to change already sent messages, a follow up edit message can be send. A COI-compatible client SHOULD only show the latest edited message, but it MAY make the history available. An edit message MAY be empty, this means that the original message should be shown as deleted. A COI client SHOULD visualize the edited state of a message.

An edit message MUST contain set the "Chat-Content" header field to "edit" with the message-ID of the large message set as a "reference" parameter in the same header field, for example:

```
 Chat-Content: ammend;
  reference=<coi$434571BC.8070702@sample.com>
```

If a message should be deleted, then the COI client MUST set the "Chat-Content" header field to "delete" and the reference parameter MUST be defined:

```
 Chat-Content: collapse;
  reference=<coi$434571BC.8070702@sample.com>
```

A message can be edited and/or deleted several times and COI client SHOULD only show the latest version. An edit and delete message MAY contain a read receipt request, which SHOULD be processed normally. To not create a sense of false security, it is RECOMMENDED that clients show the history of edited and delete messages. A deleted message is RECOMMENDED to be shown as "this message has been deleted".

A COI client MUST ensure that the referenced message is coming from the same sender as the original message.

Multiple edits always reference the last edited message, not the original message.

If a the referenced message is not found, e.g. because it is deleted, then the only new message SHOULD be shown.

If a client decides not to support edit and delete message, it just shows all messages in their chronological order.

Example 1: A normal message is being sent with a typo:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>

My dear friend, this sdkalsk.
```

Example 2: The above message is being edited with a follow up message:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S232371BC.2232&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Chat-Content: edit;
 reference=<coi$S2571BC.2878&8@sample.com
MIME-Version: 1.0

My dear friend, this is a message to you!
```

Example 3: The above message is being deleted with a follow up message:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject:
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$KJ23928L.298239&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Chat-Content: collapsed;
 reference=<coi$S2571BC.2878&8@sample.com>
MIME-Version: 1.0

(original message marked as deleted)
```

Read Receipts / Disposition Notification Reports

COI-compatible clients can request and send read notifications by following the disposition notification standards defined RFC 8098 and RFC 3503.

Requesting Read Receipts

You can can request read receipts by adding the "Disposition-Notification-To" header field with a value of your email address, for example:

```
Disposition-Notification-To: Me Myself <me@sample.com>
```

Sending Read Receipts

If the user approves sending read requests, a COI-compatible client SHOULD send our corresponding reports when receiving a chat message that:

- contains the Disposition-Notification-To header field, and
- this value is not equals the user's email address, and
- the message has not the $MDNSent keyword, and
- the message has been shown to the user.

A COI-compatible client should then sent out a message disposition notification using the following format:

```
From: Alice <you@recipientdomain.com>
To: Bob <me@sample.com>
Subject: Chat: Read receipt
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$122B223.823202@sample.com>
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Content-Type: multipart/report; report-type=disposition-notification;
    boundary="RAA14128.773615765/recipientdomain.com"
Content-Disposition: notification
MIME-Version: 1.0
```

```
--RAA14128.773615765/recipientdomain.com
 Content-Type: message/disposition-notification

Reporting-UA: joes-pc.cs.recipientdomain.com; Foomail 97.1
Original-Recipient: rfc822;you@recipientdomain.com
Final-Recipient: rfc822;you@recipientdomain.com
Original-Message-ID: <coi$199509192301.23456@sample.com>
Disposition: manual-action/MDN-sent-manually; displayed

--RAA14128.773615765/recipientdomain.com--
```

To reduce traffic, a COI-compliant client SHOULD NOT include the original message like suggested by RFC 8098.

After sending the read receipt, the client MUST set the $MDNSent keyword for the original message.

TODO describe how to do that.

Contact Storage Messages

To synchronize contacts across devices, a COI-compatible client MAY store contacts as individual messages into the COI/CONTACTS folder. Note that on a COI-compatible IMAP server this folder is generated automatically, compare the COI server specification for details.

To keep the user in control, a COI client SHOULD ask the user for permission, before storing contacts on the IMAP server.

As there are varying needs between client implementations, only some headers are specified in this specification, these headers cannot be used to calculate the contact. Message contents can contain contact data. Client MAY choose to encrypt these messages to protect the contact data further.

The contact data itself is SHOULD to be stored VCARD data of the contact as defined in RFC 6350.

A contact message SHOULD be a multipart/mixed message with the following contents:

- One text/plain part explaining the contents, e.g. "this message contains contact information". This part SHOULD have a "Content-Disposition: inline" field.
- One application/json part that contains contact meta data. This part is RECOMMENDED to have a "Content-Disposition: attachment" field with the "filename" parameter, e,g. Content-Disposition: attachment; filename="contact.json". Additionally, this part is RECOMMENDED to set the "Chat-Content" header to "contact".
- The message MAY contain a text/vcard part containing the actual contact data. This part is RECOMMENDED to have a "Content-Disposition: attachment" field with a "filename" parameter, which is RECOMMENDED to be "contact.vcf".

The "Chat-Content" header field of the JSON part MUST be set to "contact".

A COI client SHOULD use the tokens information to identify if a contact already exists before storing a new contact message.

Basic Contact Management Information

The JSON part of the contact message MUST contain the following keys:

- "coi-version" that is set to 1.
- "content" that is set to"contact".

The JSON data MAY also contain the following keys:

- "is_me" is set to true for the user's own profile. is_me is false by default
- "is_visible" which can be set to false and is true by default. This can be useful to hide contacts that have been blocked by a user. In this case the client can still show which contacts are blocked, but the contact itself is otherwise not interesting to the user anymore.
- "status" which contains the current user-generated status of the contact.
- "groups" with an array of any contact-groups a contact is associated with. Predefined groups are "partner", "family", "friend", "favorite", and "work".
- "notification" object which contains the following keys:
    - "is_enabled": true or false - defaults to true. Defines if notifications are enabled for this users.
    - "types" which is an array of strings that can contain "sound", "banner" or other client specific keywords that define the notification type that should be shown.
    - "sound" which defines the notification sound that should be used in a client specific way. RECOMMENDED values are "partner", "family", "friend", "favorite", "work" and "other",
    - "filters" criteria when a notification should be shown. When missing and is_enabled is true, it is RECOMMENDED that a notification is shown for any new user generated message. Possible values are
        - "message" for any user generated message excluding read receipts, contact updates, etc,
        - "mention" for messages that contain a mention of this user, or
        - a client specific value.

Contact Headers

To efficiently process contact messages on the server side, some data must be stored in the headers, specifically the "COI-Token" and "COI-Blocked" header fields that are described below.

- *From*: Contact's primary address and name. Optional, and in case contacts are encrypted it should only be a fixed string.
- *To*: Logged in user's primary address for this contact. Optional, and in case contacts are encrypted it should only be a fixed string.
- *Subject*: "Chat: Contact" or some other informative text.
- *Date*: Timestamp when the contact was last intentionally modified (which can be different from received-time, which is when it was uploaded to server)
- *COI-Token*: See below
- *COI-Blocked*: yes - COI server should block all messages coming from this contact. This header MUST NOT exist if the contact is not blocked.

COI-Token Header

All the token information required by the COI server is available in the COI-Token message headers. There can be multiple such headers, because each contact can have multiple aliases. The COI-Token format contains hyphen-separated list of fields:

- *Version ID* = '1'. This can be increase to completely change the format or e.g. to switch to a different hash algorithm.
- *Token string*, which can contain any characters valid in the header, except '-'
- *Base64 encoded SHA256(from + ':' + to)* of the normalized From and To addresses. Normalization means lowercasing the address, and in case of EAI addresses also converting it into UTF-8. (FIXME: Not sure of this exactly? And in theory the localpart could be case-sensitive, although practically it never is.)
- *Token creation* UNIX timestamp in hex
- *Token validity* time in seconds in hex
- Optional future extension fields, which can be ignored if they aren't understood

Blocking Contacts

To mark a contact as blocked a COI client MUST set the "COI-Blocked" header field of the corresponding contact storage message. The actual content of this header field is ignored but is RECOMMENDED to be set to "yes". When a contact is blocked, a COI client SHOULD move any incoming messages to the "Blocked" folder, unless the IMAP server supports the COI capability, in which case the server does that.

Before blocking a contact, the client SHOULD check the last received message of that contact for a DKIM header. In case there is no DKIM header, the client SHOULD warn the user about the possibility, that the sender address might be spoofed.

Contact Storage Message Example

The following example shows a complete, unencrypted contact message. In case the client chooses to encrypt the actual content data, the corresponding key as defined above will be on the same level as the "content" key.

```
From: Simon Tester <simon@coi-dev.org>
To: alice@example.com
COI-Token: 1-tbudgBSg+bHWHiHnlteNzN8TUvI80ygS9IULh4rklEw=-Js0c9gM0xZ1TMWGBQhPwiATiXOiPKRy4Ow1SmZnZu2k=-5c4f90a2-
1e13380
Subject: Chat: Contact
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$232888.2323@sample.com>
Chat-Version: 1.0
Content-Type: multipart/mixed;
 boundary="RAA14128.773615765/recipientdomain.com"
MIME-Version: 1.0

--RAA14128.773615765/recipientdomain.com
Content-Type: text/plain
Content-Disposition: inline

This message contains a contact, do not delete.

--RAA14128.773615765/recipientdomain.com
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="contact.json"
Chat-Content: contact

{
  "coi-version": 1,

  "content": "contact",

  "is_me": false,

  "is_visible": true,

  "status": "idling around...",

  "groups": ["friends", "favorite"],

  "last_time_contact_sent": "Sun, 3 Dec 2019 20:53:37 +0100",
```

```
  "notification": {
    "enabled": true,
    "types": ["banner", "sound"],
    "sound": "friend"
  }
}

--RAA14128.773615765/recipientdomain.com
Content-Type: text/vcard; charset=UTF-8
Content-Disposition: attachment;
 filename="contact.vcf"

BEGIN:VCARD
VERSION:4.0
EMAIL;TYPE=work:stester@coi-dev.org
EMAIL;PREF=1:simon@coi.me
FN:Simon Tester
N:Tester;Simon;;;ing. jr,M.Sc.
BDAY:--0203
ANNIVERSARY:20090808T1430-0500
GENDER:M
LANG;PREF=1:fr
PHOTO:
 AQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTI05ldHNjYXBlIENvbW11bm
 ljYXRpb25zIENvcnBvcmF0aW9uMRwwGgYDVQQLExNJbmZvcm1hdGlvbiBTeXN0
 <...remainder of base64-encoded data...>
END:VCARD

--RAA14128.773615765/recipientdomain.com--
```

Contact Information Message

COI clients use contact information messages to inform communication partners of their users about their profile. The initial, user-generated message in a 1:1 conversation SHOULD be a contact information message. When users change their profile ("is_me": true), a COI client MAY send this updated profile to all active contacts. What is deemed active and if a COI client differentiates between contacts is a client specific decision. It is RECOMMENDED that all contacts from which chat messages have been received in the last 30 days are deemed active.

The initial contact information message MUST set the "Content-Type" to "multipart/mixed" and MUST contain at least one text/plain or multipart/alternative, one application/json and one text/vcard part. For the text/json part of the message the "Chat-Content" header MUST be set to "contactinfo". It is RECOMMENDED to send contact information messages in an end-to-end encrypted form. When sending a contact configuration message to several recipients, the message recipients SHOULD NOT be specified in the "To" header field unless the message is sent to a group conversation. The JSON part only contains the "status" data field in this current specification, along with "coi-version" and "content" fields.

The application/json and text/vcard parts SHOULD have a "Content-Disposition" of "attachment". They SHOULD specify the "filename" parameter, which is RECOMMENDED to be "contact.json" and "contact.vcf" respectively.

Subsequent update contact information messages MAY skip the text/vcard part when only the status of a user has changed.

To reduce unnecessary traffic, it is RECOMMENDED that a COI client keeps track which contacts have received the contact information message already. COI clients MAY use a client specific key in the JSON construct of a contact storage message, e.g. "com_example_myawesesomeclient_TimeContactSent": "2019-03-19T07:22Z" .

Example:

```
From: Alice <me@sample.com>
To: you@recipientdomain.com
Subject: Chat: Hello my ...
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$238788.AB29702@sample.com>
Content-Type: multipart/mixed;
 boundary="SDKJK2233.23278827/sample.com"
Chat-Version: 1.0
MIME-Version: 1.0
--SDKJK2233.23278827/sample.com
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline

Hello my friend, this is my first COI message!
--SDKJK2233.23278827/sample.com
Content-Type: application/json
Content-Disposition: attachment; charset=UTF-8
 filename="contact.json"
Chat-Content: contactinfo

{
 "coi-version": 1,
 "content": "contact",
 "status": "working from home..."
}
```

```
--SDKJK2233.23278827/sample.com
Content-Type: text/vcard; charset=UTF-8
Content-Disposition: attachment;
 filename="contact.vcf"

BEGIN:VCARD
VERSION:4.0
EMAIL;PREF=1:me@sample.com
FN:Alice Angel
N:Angel;Alice;;;
BDAY:--1209
GENDER:M
LANG;PREF=1:en
PHOTO:
 AQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBAoTI05ldHNjYXBlIENvbW11bm
 ljYXRpb25zIENvcnBvcmF0aW9uMRwwGgYDVQQLExNJbmZvcm1hdGlvbiBTeXN0
 <...remainder of base64-encoded data...>
END:VCARD

--SDKJK2233.23278827/sample.com--
```

Note that a contact update might also only contain a status update:

TODO consider contact update message, so that COI clients discard the text/plain part

```
From: Me Myself <me@sample.com>
Subject: Chat: Contact
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$34343.ASDS9702@sample.com>
Content-Type: multipart/mixed;
 boundary="23982989.SKDJKJK@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0

--SDKJK2233.23278827/sample.com
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline

here's my new status!

--SDKJK2233.23278827/sample.com
Content-Type: application/json
Content-Disposition: attachment; charset=UTF-8
 filename="contact.json"
Chat-Content: contactinfo

{
 "coi-version": 1,
 "content": "contact",
 "status": "here's my new status!"
}

--SDKJK2233.23278827/sample.com--
```

Requesting Contact Information

When a COI client has lost information about a contact and cannot restore the information neither from the COI/CONTACTS IMAP folder nor from the message history, it MAY request the contact information by setting the Chat-Content header field to "contactrequest". The receiving COI client SHOULD send the contact information to the requesting party. It is RECOMMENDED to ask the recipient for approval before sending the contact information.

Example:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject: Chat
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Chat-Content: contactrequest
MIME-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>

(please send contact information)
```

Group Messages

Sending messages to a group is simple in principle: just send message to several participants that are defined in the "To" field.

Each group has an assigned ID. The group ID MUST only contain alphanumeric , minus and underline characters: [0-9A-Za-z_-]+. A group ID MUST be case-sensitive and MUST contain at least 12 characters.

To identify groups in replies from non-COI-compliant clients, the group ID is embedded into the message ID in the format "coi$group.<groupd ID>.<domain unique value>@<domain>", e.g.

```
Message-ID: <coi$group.4321dcba1234.ABSKDJK23223.293892839@example.com>
```

Groups are identified by parsing this Message-ID or - in the case of replies from non-COI-compliant clients - the first message-ID in the "Reference" header field.

It is RECOMMENDED that the group can be managed like email, i.e. any user can add or remove anyone from the list of recipients by changing either To or CC. Any group member SHOULD be able to change the group name, description and avatar.

To enable group avatars, allow participants to leave a group, support to manage groups, etc., COI clients use additional, specifically formatted message parts that are described below.

Sending a Chat Message to a Group

A COI client sends a message to a group by adding several contacts to the "To" header field. If a group-name exists, a COI client SHOULD specify the group name with the "Subject" header field and the group participants in the "To" header field. To not accidentally spread user-specific nicknames for contacts, COI clients SHOULD either use names authorized by the participants or use the email addresses without names in the "To" header field.

Example for a message to a group called "My Crew" and a group ID of "4321dcba1234":

```
From: Me Myself <me@sample.com>
To: alice@example.com; bob@example.com
Subject: My Crew
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$group.4321dcba1234.S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$group.4321dcba1234.434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>

Hi gang, hope you're doing fine!
```

Replying to a Group Message

When replying to a group message, COI clients MUST reply to all group participants by default, similarly to the "reply all" function of most email clients. COI clients MAY offer switching to a 1:1 conversation based on a group participant's message or from the group participant's list.

Group Storage Message

To store group information on the IMAP server, a group storage message is being used. A group consists of the following elements:

- A group name.
- The ID of the group, which uses the same structure like a message-ID.
- A list of all participants including the current user, with each participant having:
    - an "email_hash" field with the SHA3-256 hash of the participant's email address, this acts as a pointer to the actual contact.
    - the optional field "public_key", which contains the public open PGP key of that participant.
- An optional avatar for the group.
- An optional description of the group.

A group storage message is similarly structured as a contact message. The "Chat-Content" header of the message part MUST be set to "group":

```
Chat-Content: group
```

The JSON part of the message MUST have the following keys:

- "coi-version" with a value of "1".
- "content" with a value of "group".

The JSON part of the message MAY contain the following keys:

- "notification" with the same structure as for contacts.
- "group" object with non-encrypted group data:
    - "participants" as described above
    - "name" with the name of the group,
    - "id" with the GUID of the group.
    - "description" with an optional description of the group,
    - the optional "avatar" object
        - the type of the avatar like "image/webp".
        - the data of the avatar of of the group which is base64 encoded according to RFC 4648.

If the group contents should be stored encrypted, a client-specific key that ends in "_group" should be used, compare the encrypted contacts section above. Example:

```
"com_example_myawesomecoiclient_group": "<encrypted group data goes here>"
```

Group storage messages SHOULD be stored in the COI/CONTACTS folder, unless the ENABLE COI IMAP command returns a differently configured folder prefix. Compare the COI server spec for details.

Example for a non-encrypted JSON-part of a group contact message:

```
{
    "coi-version": 1,
    "content": "group",
    {
        "group": {
            "participants": [
                { "email_hash": "<SHA3-256 hash of the email address>, "public_key": "<base64 encoded key data>"
},
                { "email_hash": "djk232u8uKJHKJK@#2k32kjk2",           "public_key": "23kjkjKK...23kj23KJKH" },
                { "email_hash": "sdhj923kjkjKSKDHJiuiUIKJ<J2223",      "public_key": "jnhjhjJj23989KJKJk" },
                { "email_hash": "ssdlkwsl2023k9823kjkjKSKDHJ23230903", "public_key": "sad2320oslkdl" },
            ],
            "name": "COI Discussion",
            "description": "",
            "avatar": {
                "type": "image/jpeg",
                "data": "<base64 encoded image data>"
            },
            "id": "as989KJK287JHJ822jhj828782iNBN28768"

        },
        "notification": {
            "enabled": true,
            "types": ["banner"],
            "filter": "mention"
        }
    }
}
```

Initial Group Message: Group Creation Message

When creating a new group, COI clients MUST include the group description in the first user generated message. As with any other conversation, a COI client SHOULD also include the user's contact information message part in the same first user generated message, if the contact information has not yet been distributed to all the group participants beforehand.

The initial group message MUST set the "Content-Type" to "multipart/mixed". The "Chat-Content" header of the the group JSON part MUST be set to "groupinfo". The JSON part contains the fields "name", "description" and "avatar" like in a group contact message. Additionally, the group participants are defined like in the group storage message, however this time they also include the email address of the users. The COI client SHOULD add the public PGP keys of the participants in the field "public_key".

In summary, an initial group message contains the following parts:

- original message part, e.g. text/plain or image/webp
- application/json with Chat-Content: groupinfo, which contains the group data
- application/json with Chat-Content: contactinfo, which contains the meta information and status about the sender. This part is only present when the contact data has not yet been sent to all group participants.
- text/vcard with the actual contact data. This part is RECOMMENDED to be only present when the contact data has not yet been sent to all group participants.

Example:

```
From: Alice <alice@example.com>
To: My Crew: bob@example.com,carrol@example.com,dean@example.com;
Subject: Chat: Group creation
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$group.1234abcd4321.238788.AB29702@example.com>
Content-Type: multipart/mixed;
 boundary="A22090S.123213/example.com"
Chat-Version: 1.0
MIME-Version: 1.0

--A22090S.123213/example.com
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline

Hello folks, let's chat!

--A22090S.123213/example.com
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="groupinfo.json"
Chat-Content: groupinfo
Chat-GroupInfoSignature: jkh213k2189821hjksadhjdnbasbndjashks
 9283928k23mjk2jj8sdj2b3jda6sj2b

{
  "coi-version": 1,
  "content": "group",
  "group": {
    "name": "My Crew",
    "description": "",
    "avatar": {
      "type": "image/jpeg",
      "data": "<base64 encoded image data>"
    },
    "id": "1234abcd4321",
    "participants": [
                { "email": "alice@example.com",  "email_hash": "<SHA3-256 hash of the email address>,
"public_key": "<base64 encoded key data>", "is_admin": true },
                { "email": "bob@example.com",    "email_hash":
"djk232u8uKJHKJK@#2k32kjk2",           "public_key": "23kjkjKK...23kj23KJKH" },
                { "email": "carrol@example.com", "email_hash":
"sdhj923kjkjKSKDHJiuiUIKJ<J2223",      "public_key": "jnhjhjJj23989KJKJk" },
                { "email": "dean@example.com",   "email_hash": "ssdlkwsl2023k9823kjkjKSKDHJ23230903",
"public_key": "sad2320oslkdl" },
    ]
  }
}

--A22090S.123213/example.com
Content-Type: application/json; charset=UTF-8;
 name="contact.json"
Content-Disposition: attachment
Chat-Content: contactinfo

{
 "coi-version": 1,
 "content": "contact",
 "status": "this is my status :-)"
}

--A22090S.123213/example.com
Content-Type: text/vcard; charset=UTF-8;
 name="contact.vcf"
Content-Disposition: attachment;
 filename="contact.vcf"

BEGIN:VCARD
VERSION:4.0
EMAIL;PREF=1:alice@example.com
FN:Alice Angel
N:Angel;Alice;;;
BDAY:--1209
GENDER:F
LANG;PREF=1:en
END:VCARD

--A22090S.123213/example.com--
```

Leaving a Group: Group Participant Left Message

Any user in a group can leave a group chat. When doing so, a COI client MUST send a group participant left message. The "Chat-Content" header field MUST be set to "groupleft". A COI client SHOULD include a short description about what happened. A COI client MUST update the internal represenation of the group and remove the leaving sender of a a groupleft message.

A COI client SHOULD also send a groupleft message to a recipient that has been removed from a group.

When the only admin leaves a group or when the current user has been removed from a group, that group MAY be marked as deleted/inactive by a COI client. It is RECOMMENDED that apart from viewing the message history and deleting the group locally, no other interaction is possible from that point onward.

Note that a COI client user MAY receive further group messages after being removed from a group, for example when another group member was offline in between and had messages for that group waiting in a queue.

Example:

```
From: Bob Barr <bob@example.com>
To: alice@example.com, carrol@example.com, dean@example.com;
Subject: My Crew
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$group.1234abcd4321.Asd23212.A2328@example.com>
Reference: <coi$group.1234abcd4321.238788.AB29702@example.com>
Content-Type: text/plain; charset=UTF-8
Chat-Version: 1.0
Chat-Content: groupleft
MIME-Version: 1.0

left group "My Crew": Bob Barr
```

Group Update Message

Any group member can update a group, i.e. change the name, description and/or avatar.

The format of the group update message part is the same as for the group creation message.

TODO describe what happens when non-COI compliant clients change the recipients list, specifically when replying to an older message.

Example:

```
From: Alice <alice@example.com>
To: bob@example.com; carrol@example.com; dean@example.com
Subject: My Crew
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$group.1234abcd4321.238788.AB29702@example.com>
Content-Type: multipart/mixed;
 boundary="A22090S.123213/example.com"
Chat-Version: 1.0
MIME-Version: 1.0

--A22090S.123213/example.com
Content-Type: text/plain; charset=UTF-8
Content-Disposition: inline

Updated group avatar

--A22090S.123213/example.com
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="groupinfo.json"
Chat-Content: groupinfo
Chat-GroupInfoSignature: 2j1312kSDSD3k2DFDFFsadhjdnbasbndjashks
 92232KJKJK28k23mjk2jjDFDFF2b2378JHJ2837JHJHJL2398KBNBvb23JHJjj

{
  "coi-version": 1,
  "content": "group",
  "group": {
    "name": "My Crew",
    "description": "",
    "avatar": {
      "type": "image/jpeg",
      "data": "<base64 encoded image data>"
    },
    "id": "1234abcd4321",
    "participants": [
      { "email": "alice@example.com", "email_hash": "<SHA3-256 hash of the email address>, "public_key": "<base64
encoded key data>" },
      { "email": "bob@example.com", "email_hash": "djk232u8uKJHKJK@#2k32kjk2", "public_key": "23kjkjKK...
23kj23KJKH" },
      { "email": "carrol@example.com", "email_hash": "sdhj923kjkjKSKDHJiuiUIKJ<J2223", "public_key":
"jnhjhjJj23989KJKJk" },
      { "email": "dean@example.com", "email_hash": "ssdlkwsl2023k9823kjkjKSKDHJ23230903", "public_key":
"sad2320oslkdl" },
    ]
  }
}

--A22090S.123213/example.com--
```

Group Message Encryption

If the group conversation should be encrypted, the group creator needs to include the "public_key" field for each group participant in either the group creation message. If the public keys are known for all group participants, a COI client SHOULD encrypt messages for the group by default.

Request Group Information

When a COI client lost the group information and cannot restore it neither from the COI/CONTACTS IMAP folder nor from the message history, then it MAY create the group definition based on the group-ID, group-name and participants taken from the "To" header field. It is RECOMMENDED to request the group information. This is done by setting the "Chat-Content" header field to "grouprequested" and sending a message to all group members.

A receiving COI client of an admin participant in that group SHOULD send the information to the requesting party automatically.

Example:

```
From: Alice <alice@example.com>
To: bob@example.com, carrol@example.com, dean@example.com;
Subject: My Crew
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Chat-Content: grouprequested
MIME-Version: 1.0
Disposition-Notification-To: Alice <alice@example.com>

(please send group information)
```

Group Messaging and Mailing Lists

A mailing list discussion is from a clients view like a 1:1 chat with one person. Sometimes, users answer to both to the sender of a message and to the mailing list. It is RECOMMENDED that a COI client detects such duplicate messages and only shows the message list message, not additionally a 1:1 chat with the sender. This can be detected by looking at the message-ID given in the "Reference" or "Reply-To" header fields.

Mentions

It is RECOMMENDED that COI clients allow to mention users and that mentions for the current users are highlighted.

Mentions follow the scheme "?NAME"?:EMAILADDRESS, e.g. Gabe:gabe+private@example.com or "Gabe Lastname":gabe+private@example.com.

It is RECOMMENDED that COI clients only show the name-part of the mention text, not the full mention with the email address.

Location Messages

Sometimes it may be useful to share your location with other users. COI clients SHOULD support sending and displaying location messages.

TODO describe message format with application/vnd.google-earth.kml+xml mime type attachment.

Pinned Messages

To allow highlighting important messages, any chat message may be pinned. To achieve this, the "Content-Disposition" header field is set to "pinned". Optionally, a "timeout" parameters can be given, that contains a datetime until the message should be pinned.

COI clients SHOULD keep a pinned message in visible view area until:

- The specified timeout occurred, or
- a new pinned message arrived in a conversation, or
- the user locally unpinned a message.

Example:

```
From: Me Myself <me@sample.com>
To: You <you@recipientdomain.com>
Subject: Chat: Hello COI...
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$S2571BC.2878&8@sample.com>
Content-Type: text/plain; charset=UTF-8
Content-Disposition: pinned;
    timeout=Tue, 5 Dec 2019 0:00:00 +0100
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>

Hello COI world, this message is pinned!
```

Poll Messages

Polls are an integral part of social communication, e.g. for finding a suitable date or selecting a preferred option. To provide support for this interaction, COI clients SHOULD support displaying and voting poll messages.

Poll Message

A COI client MUST set the "content" parameter of the "Chat-Version" header field to "poll". To provide non-COI-compliant clients a possibility to display content, a poll message MUST be a multipart/alternative message with at least one text/plain and/or text/html part and one application/json part present. A poll message is RECOMMENDED to be pinned.

A poll JSON object consists of the following elements:

- "coi-version" which is set to 1.
- "content" which is set to "poll".
- "poll" which contains the actual poll data:
    - "title" which contains the poll's title.
    - "options" which contains an array of options. At least two options have to be present.
        - Each option contains a description element.
    - An optional "show-intermediate-results" element which defaults to true. If set to false, a COI client SHOULD NOT show the intermediate result of a poll when a user votes.
    - An optional "data" element that contains binary data like an image or an animation that is relevant to the poll in base64 encoding according to RFC 4648. If a "data" element is present, a "data-type" element MUST be present and set to the corresponding MIME type. COI clients MUST support all image, video and audio types as defined in the binary messages section above.
    - An optional "closed" element that is false by default. When a poll is closed, COI clients MUST NOT allow to vote for it.

Example:

```
From: Me Myself <me@sample.com>
To: My Crew: you@recipientdomain.com, carrol@otherdomain.com, dareen@somewhere.com;
Subject: Chat: poll - Which movie?
Date: Mon, 4 Dec 2019 15:51:37 +0100
Message-ID: <coi$group.23298982923929.238788.AB29702@sample.com>
Content-Type: multipart/alternative;
 boundary="unique-boundary-1"
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
Disposition-Notification-To: Me Myself <me@sample.com>
Content-Disposition: pinned;
 timeout=Tue, 5 Dec 2019 0:00:00 +0100
MIME-Version: 1.0

--unique-boundary-1
Content-Type: text/html; charset=UTF-8

<html><body>
<p>Which movie: &quot;Avengers&quot;, &quot;Matrix&quot;, &quot;Aliens&quot;?</p>
<p><i>This message is a chat message - consider using <a href="https://myawesomecoiapp.com">my awesome COI app<
/a> for best experience!</i></p>
</body></html>

--unique-boundary-1
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="poll.json"
Chat-Content: poll

{
 "coi-version": 1,
 "content": "poll",
 "poll": {
 "title": "Which movie?",
 "options": [ { "description": "Avengers" }, { "description": "Matrix" }, { "description": "Aliens" } ],
 "show-intermediate-results": false,
 "data-type": "image/webp",
 "data": "SKJ098788787K878J878...rest of base64 encoded WebP image"
 }
}

--unique-boundary-1--
```

Poll Vote Message

COI clients SHOULD support poll vote messages. Each recipient of a poll message can vote multiple times, in this case only the latest vote message counts. A poll vote message MUST be a multipart/alternative message with at least one text/plain or text/html part explaining the vote and one "application /json" part containing the actual vote data. The "Chat-Content" header field of the JSON part MUST be set to "pollvote" and the "reference" parameter to the poll's message-ID. When a poll has been closed, a COI client MUST NOT allow to vote for it - instead the final result SHOULD be shown.

During voting a COI client is RECOMMENDED to show the intermediate votes results, unless the poll's "show-intermediate-results" element is set to false. To achieve this, a COI client is RECOMMENDED to count the votes of each conversation participant's last vote message. If a user has cast a vote before, a COI client SHOULD visualize the previous votes and pre-select those choices.

A poll vote JSON object consists of the following parts:

- "coi-version" which is set to 1.
- "content" which is set to "pollvote".
- "votes" is an array with the indices of the voted options, first index is 0.

Example:

```
From: Me Myself <me@sample.com>
To: My Crew: you@recipientdomain.com, carrol@otherdomain.com, dareen@somewhere.com;
Subject: Chat: vote - Which movie?
Date: Mon, 4 Dec 2019 16:51:37 +0100
Message-ID: <coi$group.23298982923929.21382788.2932UJH@sample.com>
Content-Type: multipart/alternative;
 boundary="unique-boundary-1"
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0

--unique-boundary-1
Content-Type: text/html; charset=UTF-8

<html><body>
<p>Which movie: voted for &quot;Matrix&quot; and &quot;Aliens&quot;.</p>
<p><i>This message is a chat message - consider using <a href="https://myawesomecoiapp.com">my
awesome COI app</a> for best experience!</i></p>
</body></html>

--unique-boundary-1
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="poll.json"
Chat-Content: pollvote;
 reference=<coi$group.23298982923929.238788.AB29702@sample.com>

{
 "coi-version": 1,
 "content": "pollvote",
 "votes": [1,2]
}

--unique-boundary-1--
```

Poll Closed Message

The author of a poll can close a poll. A COI client marks a poll as closed when such a poll closing message has been received and the sender is the same as the original poll.

A poll vote message MUST be a multipart/alternative message with at least one text/plain or text/html part explaining the vote and one "application/json" part containing the actual vote data. The "Chat-Content" header field of the JSON part MUST be set to "pollclose" and the "reference" parameter to the poll's message-ID. When a poll has been closed, a COI client MUST NOT allow to vote for it - instead the final result SHOULD be shown. The referenced poll message should be unpinned unless the "pinned" element is set to "true". During closing of the poll, a COI client SHOULD allow the user to select an option.

A poll close JSON object consists of the following parts:

- "coi-version" which is set to 1.
- "content" which is set to "pollclose".
- "result" which is an object with the following elements:
    - "index" is set to the index of the chosen result. Note that any option can be selected.
    - optionally "pinned" which defaults to false. When set to true, the original message is kept being pinned. By default the original poll message is being unpinned with the pollclose signal.

Example:

```
From: Me Myself <me@sample.com>
To: My Crew: you@recipientdomain.com, carrol@otherdomain.com, dareen@somewhere.com;
Subject: Chat: vote - Which movie?
Date: Mon, 4 Dec 2019 16:51:37 +0100
Message-ID: <coi$group.23298982923929.21382788.2932UJH@sample.com>
Content-Type: multipart/alternative;
 boundary="unique-boundary-1"
Reference: <coi$434571BC.89A707D2@sample.com>
Chat-Version: 1.0
MIME-Version: 1.0

--unique-boundary-1
Content-Type: text/html; charset=UTF-8

<html><body>
<p>Which movie: voted for &amp;quot;Matrix&amp;quot; and &amp;quot;Aliens&amp;quot;.</p>
<p><i>This message is a chat message - consider using <a href="https://myawesomecoiapp.com">my awesome COI app<
/a> for best experience!</i></p>
</body></html>

--unique-boundary-1
Content-Type: application/json; charset=UTF-8
Content-Disposition: attachment;
 filename="poll.json"
Chat-Content: pollclose;
 reference=<coi$group.23298982923929.238788.AB29702@sample.com>

{
 "coi-version": 1,
 "content": "pollclose",
 "result": {
   "index": 2,
   "pinned": true
 }
}

--unique-boundary-1--
```

Poll Deleted and Poll Edited Messages

A poll message can be edited and deleted like a normal COI chat message. A COI client SHOULD discard any votes up to the editing message will be in that case.

Extension Messages

COI clients MAY introduce client specific messages. Those messages SHOULD always include a text/plain, text/html or binary inline part explaining the contents for users of non COI compliant clients. COI clients MAY use the "Chat-Content" header field for their data message parts. To ensure uniqueness of "Chat-Content" header values, clients MUST start the value with a client-specific name. A reverse domain for the value start is RECOMMENDED, for example:

```
Chat-Content: com.awesomeclient.mycustomevent
```

Message Encryption

A COI-compliant client MAY support the Autocrypt standard to ease end to end encryption scenarios.

TODO: Consider using more secure lookup mechanisms for encryption keys. Also check for existing encryption keys before auto-generating a new encryption key set.

Blocking Users

A COI client SHOULD ignore any messages in from contacts that have set "is_blocked" to true. Such messages SHOULD be moved to the BLOCKED folder on the IMAP server. Note that a COI compliant IMAP server does this automatically.

Separate COI and Mail Messages

To keep existing email apps functional and not overflood an inbox with chat messages, COI clients that operate on non-COI-enable IMAP servers SHOULD separate COI messages from normal email messages.

COI clients SHOULD move COI messages to the COI/CHATS folder. Additionally, replies from non-COI compliant clients to COI messages SHOULD be moved to the COI/CHATS folder as well.

Replies of non-COI compliant clients can be identified as chat message by having:

- Having a reference that starts with a message ID starting with "<coi$", or
- Having an "in-reply-to" header field with a message-ID that starts with "<coi$"

For moving the message to COI/CHATS a COI client has the following options:

- COI client listens for incoming mails on INBOX either using IMAP IDLE or IMAP NOTIFY, and then moves matching chat mails manually to COI /CHATS
- pending the user's approval, a service could monitor incoming mails and do that.

After moving a message that originates from the user itself, identified either with the primary email address or a valid alias address, this message should be marked as read at the same time.

TODO describe how to do that.

Folders

When a COI client connects to a non-COI IMAP server, it might need to create several folders initially:

- COI/CHAT for storing all chat messages. This folder SHOULD be subscribed the user.
- COI/CONTACTS for storing contacts.
- BLOCKED for storing messages originating from blocked contacts.

TODO describe IMAP details

Security Considerations

The security consideration of the referenced standards also apply to this COI client specification definition.

IANA Considerations

The "pinned" value of the "Content-Disposition" header field should be registered with IANA.