COI for Beginners

Discover how to use COI

COI Principles

The principle of COI is very simple: COI uses IMAP/SMTP as the transport mechanism for a chat application.

The generic steps are as follows:

1. Establish a connection with an SMTP server (Send) and an IMAP server (Receive)
2. Establish the capabilities of the IMAP server. This determines how you communicate with it
3. Send chat messages over SMTP, receive messages over IMAP

COI Now & Soon

Today, you can use COI to communicate with other compatible messengers over existing IMAP and SMTP servers.

Soon, additional functionalities will be made available by COI compatible IMAP servers. Example for such features are automated filtering between normal mail messages and chat messages, server-side blocking of contacts, support for push notifications, channels, WebRTC and much more.

Required Knowledge

To build a COI application you need to know the following:

- How to communicate with and IMAP and SMTP server - This is usually done by using a native email library.

A basic understanding of the structure of SMTP/IMAP messages - Specifically an working knowledge of RFC 5322, but reading the COI protocol should give you most of what you need.

- A good understanding of the COI protocol in both its variants (COI over simple IMAP servers and COI over COI extended IMAP servers).
- Know how to build your desired application.

What you need

As a client developer, you have several options:

- Communicate with the email server using an IMAP library for your preferred language & platform – or even communicate directly when you feel adventurous.
- Base your work on Delta Chat Core, an MPL licensed library that abstracts away IMAP communication and that will become COI compliant.
- Base your work on an existing COI compatible app, like the cross-platform OX COI app.

Hello COI World

The following sections will show you the principles of how to create a very basic COI based application.

Background

This is not designed to show you everything but rather show you the principles involved. Although not language specific the following examples show and refer to:

- An Apple Terminal (High Sierra)
- XCode 10.1
- Swift 4.2
- MailCore2 library

You will need the following:

- Your preferred development environment and knowledge about how to use it. We will be using XCode and Swift
- An installed and tested email communication library that will let you communicate with SMTP and IMAP. We will be using MailCore2
- Access and login credentials (if needed) to an IMAP and SMTP server. We will be using public Gmail servers.

Step 1

First check your SMTP connection

This example uses an Apple Mac Terminal Session to connect to a gmail SMTP server:

```
> openssl s_client -connect smtp.gmail.com:465
CONNECTED(00000006)
depth=1 C = US, O = Google Trust Services, CN = Google Internet Authority G3
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/CN=smtp.gmail.com
[and lots more stuff]
---
220 smtp.gmail.com ESMTP w18sm33115850wmi.12 - gsmtp
> EHLO Mike [Mike can be anything]
250-smtp.gmail.com at your service, [185.91.231.176]
250-SIZE 35882577
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8

> AUTH PLAIN AE9MTM [Get AE9MTM from this "echo -ne '\00<YourEmail@gmail.com>\00<YourPassword>' | base64"]
235 2.7.0 Accepted
>> MAIL FROM: <ABC@gmail.com> [ABC@gmail.com is the from email address]
250 2.1.0 OK k26sm29754989wmi.28 - gsmtp
>> rcpt to: <XYZ@gmail.com> [XYZ@gmail.com is the recipients email address]
250 2.1.0 OK w12sm82801761wrr.23 - gsmtp
>>DATA
354 Go ahead m4sm74187003wmi.3 -gsmtp


Subject: it works
yay!
[CTRL+V+ENTER] [Normally a blank line is ok, but sometimes you need this]
. [CTRL+V+ENTER] [Normally a dot is ok, but sometimes you need this]
250 2.0.0 OK 1548086275 m4sm74187003wmi.3 - gsmtp
>> quit
221 2.0.0 closing connection m4sm74187003wmi.3 - gsmtp
```

If the recipient receives the email then you know you are ok to send. You are half way there.

Step 2

Next check your IMAP connection

This example uses an Apple Mac Terminal Session to connect to a gmail IMAP server:

> *> openssl s_client -crlf -connect imap.googlemail.com:993*
>
> *CONNECTED(00000005)*
>
> *depth=1 C = US, O = Google Trust Services, CN = Google Internet Authority G3*
>
> *verify error:num=20:unable to get local issuer certificate*
>
> *verify return:0*
>
> *---*
>
> *Certificate chain*
>
> *0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/CN=imap.googlemail.com*
>
> *[*and lots more stuff*]*
>
> *---*
>
> ** OK Gimap ready for requests from 185.91.231.176 x9mb53718556ltn*
>
> *> 1 login <Email Address> <Password>*
>
> ** CAPABILITY IMAP4rev1 UNSELECT IDLE NAMESPACE QUOTA ID XLIST CHILDREN X-GM-EXT-1 UIDPLUS COMPRESS=DEFLATE ENABLE MOVE CONDSTORE ESEARCH UTF8=ACCEPT LIST-EXTENDED LIST-STATUS LITERAL- SPECIAL-USE APPENDLIMIT=35651584*
>
> *1 OK <EMAIL ADDRESS> authenticated (Success)*
>
> *> 1 logout*
>
> ** BYE LOGOUT Requested*
>
> *1 OK 73 good day (Success)*
>
> *read:errno=0*

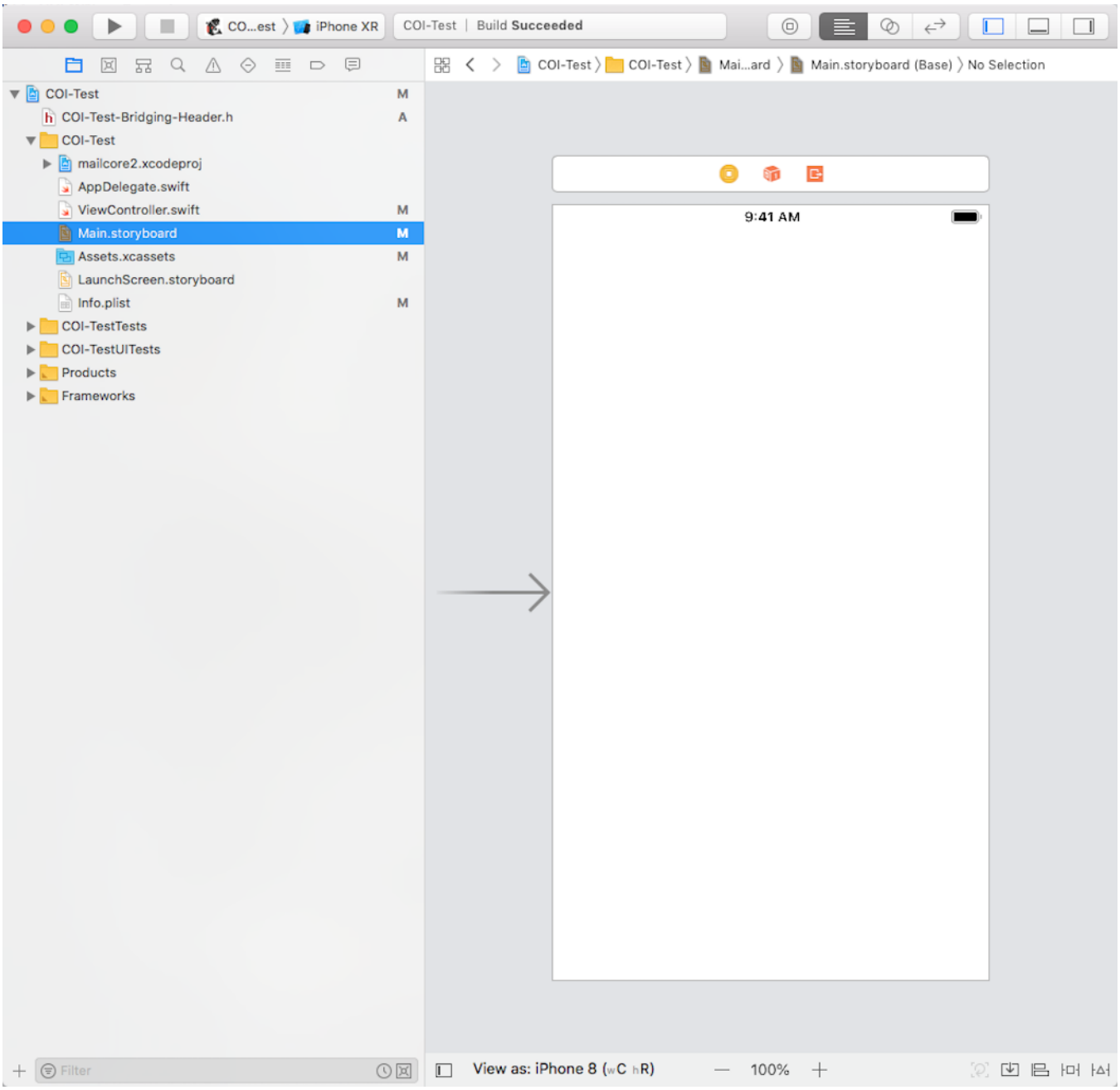
If you managed to connect and login successfully then you are good to go. Let's start to write an application.

Step 3

This bit you will have to do on your own, but basically you want to:

1. Create a simple blank application
2. Install the email communication library
3. Build, run and make sure there are no errors or warnings.

At this point you should have something like this, but in your environment and with your library installed:

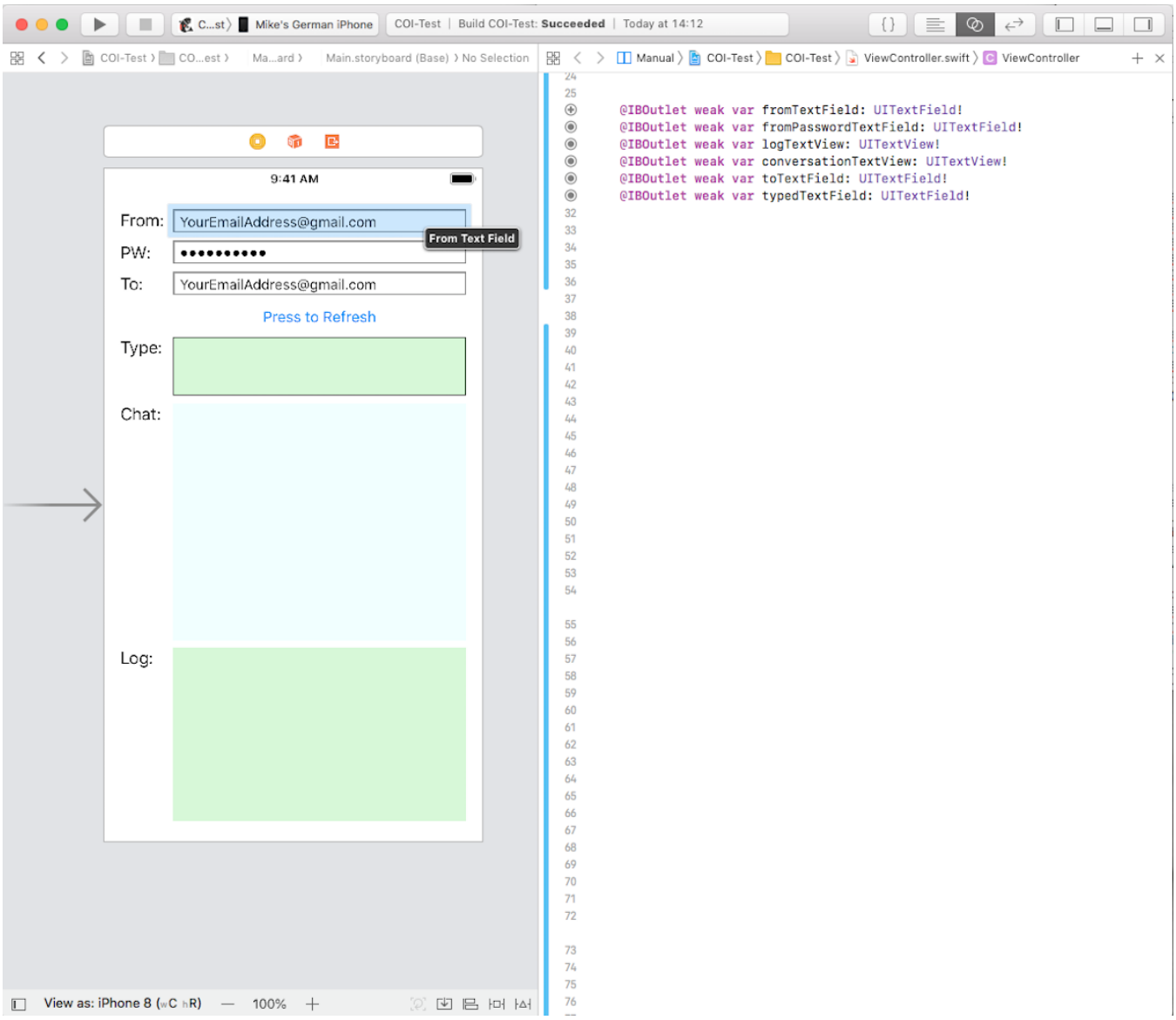


Step 4

Create the view you wish to have and link the relevant view elements to view variables.

As this is just an example it will not be beautiful, but it will be functional.

This is our example application:



Step 5

Now let's start doing email stuff.

First let's concentrate on the SMTP (sending of emails) bit.

1. Instantiate your SMTP class
2. Next populate the following class variables (this may differ depending on the library you are using):
    a. smtpSession.hostname = smtpServer
    b. smtpSession.username = fromTextField.text
    c. smtpSession.password = fromPasswordTextField.text
    d. smtpSession.port = 465
    e. smtpSession.authType = MCOAuthType.saslPlain
    f. smtpSession.connectionType = MCOConnectionType.TLS
    g. Optionally we have setup a 'connectionLogger' so that we can see what is going on:

```
smtpSession.connectionLogger = {(connectionID, type, data) in
    if data != nil {
        if let string = NSString(data: data!, encoding: String.Encoding.utf8.rawValue){
           NSLog("SMTP Connectionlogger: \(string)")
           self.logTextToScreen(text: "Connectionlogger: \(string)")
        }
        else {
           self.logTextToScreen(text: "All Ok")
        }
    }
}
```

3. Process entered text.
   In this example we have used the 'textFieldShouldReturn' delegate function to trigger the sending of a COI message.
   This function is called when the user presses return in the connected upper text field.
   When the return key is pressed we do the following:
   1. Instantiate a 'MessageBuilder' class that will build our message
   2. We then populate it with the following:
      i. From
      ii. To
      iii. Subject
      iv. MessageID (Message ID is formed as specified in the COI documentation)
      v. We have left out *Content-Type* and *MIME-Version* as MailCore2 does this for us
      vi. Chat-Version
      vii. htmlBody (the text is taken from the text field)
4. Send the processed data
   i. We then use the MessageBuild class to create the rfc822 compliant Data
   ii. Finally we use the Instantiated SMTP class to send the formatted data

Just with this step you can already see the results hitting your demo inbox.

Step 6

Now the slightly more complex bit: IMAP.

As you know SMTP is used for sending emails, but IMAP is used to receive emails.

This all starts in a similar way to SMTP:

1. Instantiate your IMAP class
2. Next populate the following class variables (this may differ depending on the library you are using):
   a. imapSession.hostname = imapServer
   b. imapSession.port = 993
   c. imapSession.username = fromTextField.text
   d. imapSession.password = fromPasswordTextField.text
   e. imapSession.connectionType = MCOConnectionType.TLS
   f. Optionally we have setup a 'connectionLogger' so that we can see what is going on:

```
imapSession.connectionLogger = {(connectionID, type, data) in
        if data != nil {
                if let string = NSString(data: data!, encoding: String.Encoding.utf8.rawValue){
                        NSLog("IMAP Connectionlogger: \(string)")
                        self.logTextToScreen(text: "Connectionlogger: \(string)")
                }
                else {
                        self.logTextToScreen(text: "All Ok")
                }
        }
}
```

3. We then use the press of a button to call a function “loadCOIMsg”
   This function will get all the COI messages and write all the unread ones to a text view
   a. To identify a COI message, in this example, we look for the subject “My COI message”
   b. We use the instantiated IMAP class function “searchExpressionOperation” to search for all emails that have a subject “My COI message”
   c. The ‘completionBlock’ receives an index list of all matches.
   d. We then use the instantiated IMAP class function “fetchMessagesOperation” to get all the headers for all the matches.
   e. The “fetchMessagesOperation” completionBlock receives an array of headers
   f. For each header we check if the message has been read.
      If not then we use the instantiated IMAP class function “fetchMessageOperation” to get the message data.
   g. The ‘completionBlock’ receives the message data for each message.
   h. In the completionBlock we then parse the message data to extract the message text itself and display it in a text view.

Full code example

Please note that this is just a demonstration example and not an elegant production or usable application. This application is an iOS example using Swift and Mailcore2:

```
//  ViewController.swift
//  COI-Test
//

import UIKit

class ViewController: UIViewController, UITextFieldDelegate, MCOHTMLRendererDelegate {

    // Constants
    let smtpServer          = "smtp.gmail.com"
    let imapServer          = "imap.googlemail.com"
    let fromUserDefault     = "<email address>"
    let fromUserPWDefault   = "<password>"
    let toUserDefault       = "<email address>"
    let COIIDPrefix         = "coi$"
    let COISubject          = "My COI message"
    let fetchRange: UInt64  = 3



    @IBOutlet weak var fromTextField: UITextField!
    @IBOutlet weak var fromPasswordTextField: UITextField!
    @IBOutlet weak var logTextView: UITextView!
    @IBOutlet weak var conversationTextView: UITextView!
    @IBOutlet weak var toTextField: UITextField!
    @IBOutlet weak var typedTextField: UITextField!

    let smtpSession = MCOSMTPSession()
    let imapSession = MCOIMAPSession()


    override func viewDidLoad() {
        super.viewDidLoad()

        // Set defaults into entry fields
        fromTextField.text = fromUserDefault
        fromPasswordTextField.text = fromUserPWDefault
        toTextField.text = toUserDefault

        // Setup Com's to GMail SMTP Server
        smtpSession.hostname = smtpServer
        smtpSession.username = fromTextField.text
        smtpSession.password = fromPasswordTextField.text
        smtpSession.port = 465
        smtpSession.authType = MCOAuthType.saslPlain
        smtpSession.connectionType = MCOConnectionType.TLS
        smtpSession.connectionLogger = {(connectionID, type, data) in
            if data != nil {
                if let string = NSString(data: data!, encoding: String.Encoding.utf8.rawValue){
```

```
                    NSLog("SMTP Connectionlogger: \(string)")
                    self.logTextToScreen(text: "Connectionlogger: \(string)")
                }
                else {
                    self.logTextToScreen(text: "All Ok")
                }
            }
        }

        // Setup Com's to GMail IMAP Server
        imapSession.hostname = imapServer
        imapSession.port = 993
        imapSession.username = fromTextField.text
        imapSession.password = fromPasswordTextField.text
        imapSession.connectionType = MCOConnectionType.TLS
        imapSession.connectionLogger = {(connectionID, type, data) in
            if data != nil {
                if let string = NSString(data: data!, encoding: String.Encoding.utf8.rawValue){
                    NSLog("IMAP Connectionlogger: \(string)")
                    self.logTextToScreen(text: "Connectionlogger: \(string)")
                }
                else {
                    self.logTextToScreen(text: "All Ok")
                }
            }
        }
    }


    @IBAction func connectUIButton(_ sender: UIButton, forEvent event: UIEvent) {
        conversationTextView.text = ""
        loadCOIMsg(folder: "INBOX")
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        let builder = MCOMessageBuilder()

        builder.header.from = MCOAddress(displayName: "COI-Demo", mailbox: fromTextField.text)
        builder.header.to = [MCOAddress(displayName: toTextField.text?.components(separatedBy: "@").first,
mailbox: toTextField.text)]
        builder.header.subject = COISubject
        builder.header.messageID = makeMessageID(fromUserEmailAddress: fromTextField.text!)

        //Romoved as MailCore2 does this automatically
        //builder.header.setExtraHeaderValue("text/plain; charset=utf-8", forName: "Content-Type")
        //builder.header.setExtraHeaderValue("1.0", forName: "MIME-Version")

        builder.header.setExtraHeaderValue("1.0", forName: "Chat-Version")

        builder.htmlBody = textField.text

        let rfc822Data = builder.data()

        let sendOperation = smtpSession.sendOperation(with: rfc822Data)
        sendOperation!.start { (error) -> Void in
            if (error != nil) {
                NSLog("Error sending email: \(String(describing: error))")
            } else {
                NSLog("Successfully sent email!")
                self.conversationTextView.text = self.conversationTextView.text + "\n" + textField.text!
                textField.text = ""
            }
        }
        textField.resignFirstResponder()
        return true
    }

    func logTextToScreen(text: String){
        DispatchQueue.main.async {
            self.logTextView.text = self.logTextView.text + "\n" + text
```

```
            if self.logTextView.text.count > 0 {
                let location = self.logTextView.text.count - 1
                let bottom = NSMakeRange(location, 1)
                self.logTextView.scrollRangeToVisible(bottom)
            }
        }
    }

    func makeMessageID(fromUserEmailAddress: String) -> String {
        var messageID: String

        let date = Date(timeIntervalSince1970: Date().timeIntervalSince1970)
        let dateFormatter = DateFormatter()
        dateFormatter.timeZone = TimeZone(abbreviation: "GMT") //Set to GMT timezone
        dateFormatter.locale = NSLocale.current
        dateFormatter.dateFormat = "yyyyMMddHHmmss"
        let strDate = dateFormatter.string(from: date)
        let domain = fromUserEmailAddress.components(separatedBy: "@").last

        messageID = COIIDPrefix + strDate + "@" + (domain ?? "")
        return messageID
    }

    func loadCOIMsg(folder: String){

        let messageSubject: String = COISubject //subject not message ID :(
        let searchExpression: MCOIMAPSearchExpression = MCOIMAPSearchExpression.searchHeader("Subject",
value: messageSubject)
        let searchOperation: MCOIMAPSearchOperation = self.imapSession.searchExpressionOperation(withFolder:
folder, expression: searchExpression)

        searchOperation.start { (error, searchIndexSet) in
            if (error != nil){
                NSLog("MG Error")
            }else{
                let requestKind: MCOIMAPMessagesRequestKind = [.fullHeaders, .extraHeaders, .flags]
                let messageOperation: MCOIMAPFetchMessagesOperation = self.imapSession.fetchMessagesOperation
(withFolder: folder, requestKind: requestKind, uids: searchIndexSet)

                messageOperation.start({ (error, messagesList, messageIndexSet) in
                    if (error != nil){
                        NSLog("MG Error")
                    }else{
                        for currentMessage in messagesList!{
                            if currentMessage.flags != MCOMessageFlag.seen{
                                let contentOperation: MCOIMAPFetchContentOperation = self.imapSession.
fetchMessageOperation(withFolder: folder, uid: currentMessage.uid)

                                contentOperation.start({ (error, messageData) in
                                    let Msg: MCOMessageParser = MCOMessageParser.init(data: messageData)
                                    self.conversationTextView.text += "\n" + Msg.plainTextBodyRendering()
                                })
                            }
                        }
                    }
                })
            }
        }
    }
}
```

What Next?

As mentioned the above is just an example exercise and not representative of how a real chat application should work. The next steps are to implement:

1. Real conversations that share Message IDs
2. Auto refresh
3. Contacts
4. Read receipts
5. Binary messages (images/audio/video)
6. Group messages
7. Other messages such as edit, preview or poll messages

After that there are many more fun things you can implement.

What Next for COI

We are in the process of defining a full RFC for COI. This extended IMAP standard will make life easier for developers, improve performance and increase what you can do.

How We Can Help

If you have questions, you can post a question on Stackoverflow using the #coi tag or add an issue on our Github project.