

Ih-Mehl – Mailclient (SS 2007)

Practical Training Telematics



Mailclient „Ih-Mehl“ – Overview



- **Project divided into four parts:**
 - Data structure classes for e-Mails and mailboxes (Christian)
 - POP3 implementation (David)
 - SMTP implementation (Salke)
 - GUI creation (Tim)
- **Used language: JAVA 1.6**
- **RFCs concerned:**
 - RFC 1939 (POP3)
 - RFC 821 (SMTP)
 - RFC 1869 (ESMTP)
 - RFC 2554 (SMTP Authentication)
 - RFC 2104 (HMAC)

Mailclient „Ih-Mehl“ – Project Organization [1 / 2]



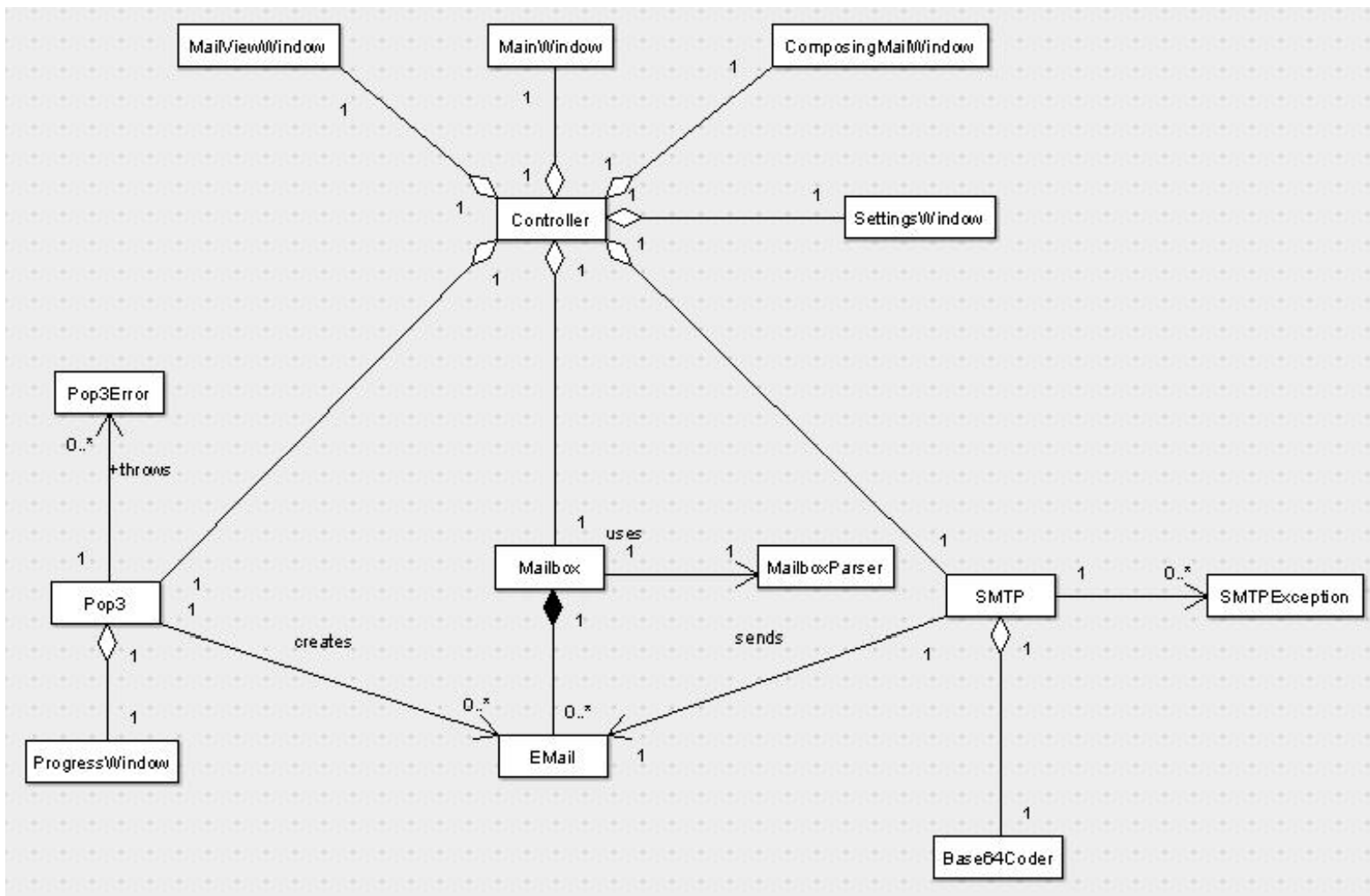
- **Project leader: Christian, but this was only an organisational position – technical decisions were made together by all project members.**
- **Used Pattern: MVC**
- **Communication...**
 - ...mostly done via IRC
 - several meetings

Mailclient „Ih-Mehl“ – Project Organization [2 / 2]



- **Initial milestones**
 - **May 13th**
 - Finished modelling, talk about how separate program parts are going to communicate with each other, create UML class diagrams
 - **June 2nd**
 - Implement basic functionality: Fetching, reading, composing and sending mails. This should have been tested coarsely as well
 - **June 17th**
 - All planned features should be implemented by now (without intensive testing). One week of testing and bug fixing after this milestone
 - **June 24th**
 - Finish testing and bug fixing. Everything should be implemented and work as intended. No more work needs to be done on the program code.
- **Milestone complied?**
 - **Yes**
 - **Yes**
 - **One week behind**
 - **One week behind**

Mailclient „Ih-Mehl“ – UML diagram (rough)



POP3 Implementation – Overview



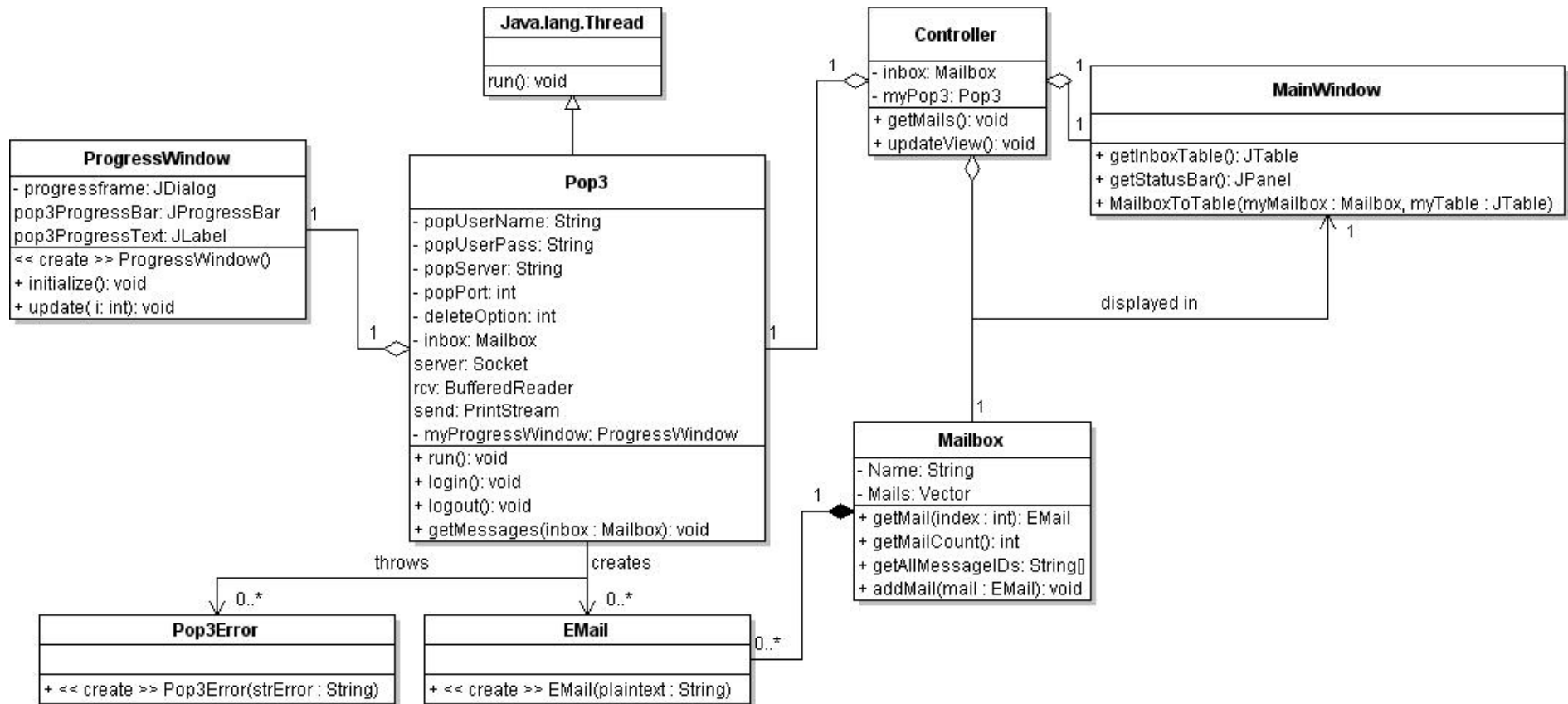
- **POP3 class implements RFC 1939**
- **Required functionality was:**
 1. Login at a POP3 server
 2. Get mails from that server
 3. Store each mail in an EMail object (-> EMail.java)
 4. Store each EMail object in the inbox Mailbox (-> Mailbox.java)
 5. Be able to delete messages on the server

POP3 Implementation – Methods



- **login() -**
tries to login at the given server
switches to transaction state mentioned in the RFC
- **getMessages() –**
fetches messages from mailaccount at the given server, if login was successful
stores every single mail in an EMail object
saves every EMail object in a Mailbox Object (inbox)
deletes messages on server (only if user has set flag to do so)
- **logout() -**
logs out after messages are fetched
terminates connection
- **run() –**
implements run() method of JAVA threading class
all methods above are called from run()

POP3 Implementation – Related Classes



POP3 Implementation – Summary



- **Functionality:**

1. Login at a POP3 server ✓
2. Get mails from that server ✓
3. Store each mail in an EMail object (-> EMail.java) ✓
4. Store each EMail object in the inbox Mailbox (-> Mailbox.java) ✓
5. Be able to delete messages on the server ✓

- **Plus new (partially not mandatory) stuff since last meeting:**

- Threading ✓
- Showing progress via GUI ✓
- Several bug fixes (duplicate mails, ...)

- **Problems:**

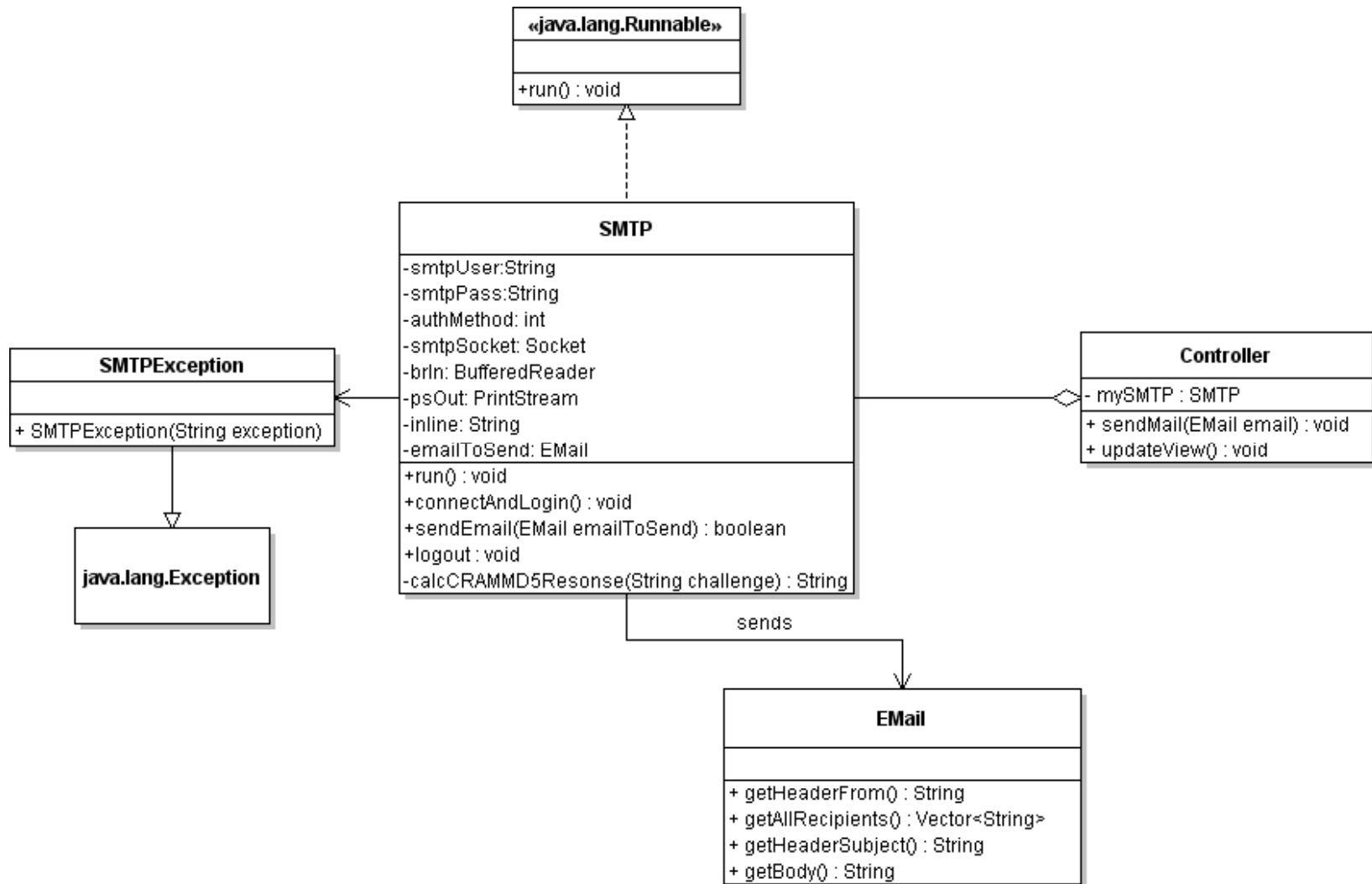
No one (even big providers like web.de) cares about correct RFC implementation ✗

SMTP Implementation - Overview



- **SMTP class implements RFC 821**
- **Required functionality was:**
 1. Sending plaintext messages (EMail objects) using SMTP
 2. Providing support for multiple recipients
 3. Providing authentication mechanisms

SMTP Implementation – Related Classes



SMTP Implementation - Methods



- **connectAndLogin()**
tries to login at the given smtp server using the auth mechanism that has been selected
- **sendEmail(EMail emailToSend)**
sends the given Email-object using the well-known smtp-commands
loops through the email's recipient-vector to provide multiple recipients
- **logout()**
closes the connection to the smtp-server
- **run()**
Implements the run-method from the Java Runnable-Interface to provide threading
All above methods are called from run()

SMTP – new code since last meeting



- **All three common SMTP-authentication mechanisms have been implemented**
- **AUTH-PLAIN and AUTH-LOGIN**
Username and password are encoded using Base64 and sent in plaintext to the smtp-Server
Only encoding, no encryption, Base64 is just a standard to encode data
- **Authentication wanted that doesn't need to send the password in plaintext:**
CRAM-MD5
Server sends a challenge encoded in Base64 to the client
Client decodes the challenge and calculates the challenge-response
Client encodes the challenge-response using Base64 and sends it back to the server
Server also calculates the challenge-response and compares the result with the client response

How's the response calculated, where's the security?

SMTP Implementation – CRAM-MD5



- **The server challenge includes a timestamp and is just valid for a defined time window**
The client has to send the response within the time window, otherwise it's declined
- **The calculation of the response is based on the MD5 hashing function**
To generate the response the client uses the HMAC-algorithm (RFC 2104) that makes multiple calls to MD5

The input for the hashing function is derived from the server's challenge and the client's username/password combination

For other details, like input padding, etc. please see the RFC or have a look at our code

- **It's impossible to reverse the output of MD5 to get the original input back**
username/password cannot be reconstructed

Controller & GUI – classoverview



- **Controller**
Initializes the application, takes care about the „control flow“, closes the application
- **ComposingMailWindow**
for mail-editing purposes, needed in order to create new mails, replying and forwarding
- **SettingsWindow**
for adjusting all options related to the POP3/SMTP settings, including loading and saving them
- **MailViewWindowComponentListener/MainWindowComponentListener**
just to react, when windows are about to be resized
- **MainWindowListener**
to perform actions, when the application terminates

Controller Class – UML Diagramm



GUI - MainWindow

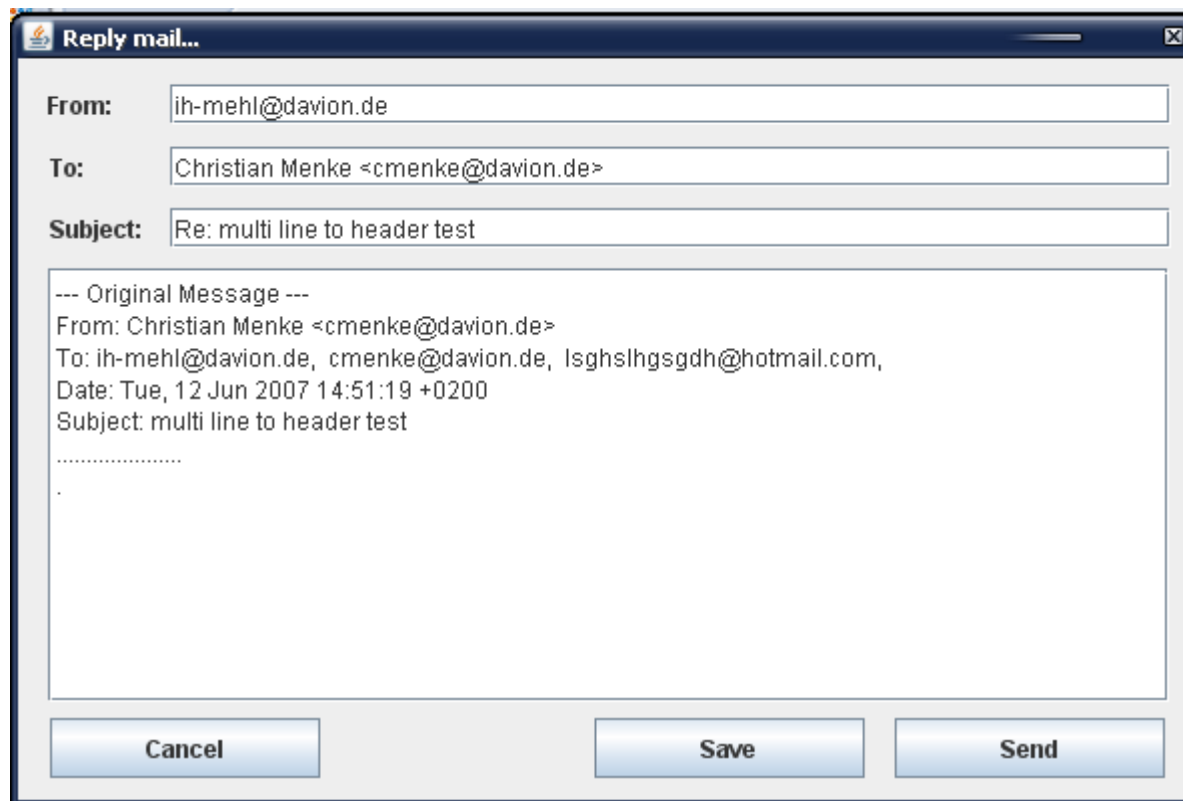


Sender	Subject	Date
Christian Menke <cmenke@davion.de>	Wer hat die Mails gelöscht?	Tue, 29 May 2007 14:27:14 +0200
"Salke Hartung" <salke.hartung@gmail.com>	Eins, Zwei, Polizei	Thu, 7 Jun 2007 20:16:57 +0200
Steffen <steffen.herbald@web.de>	test	Thu, 07 Jun 2007 20:17:16 +0200
Tim Waage <tim.waage@web.de>	FLEISCH!	Thu, 07 Jun 2007 20:17:44 +0200
ih mehl <ihmehl@hotmail.com>	testmail	Thu, 7 Jun 2007 21:17:47 +0300
"Salke Hartung" <salke.hartung@gmail.com>	test_2_empfänger	Thu, 7 Jun 2007 20:38:40 +0200
"Salke Hartung" <salke.hartung@gmail.com>	test_2_empfänger_andere_reihenfolge	Thu, 7 Jun 2007 20:39:06 +0200
Lennart Obermann <lennart.obermann@we...>	bla	Thu, 07 Jun 2007 20:59:17 +0200
Christian Menke <cmenke@davion.de>	multi line to header test	Tue, 12 Jun 2007 14:51:19 +0200
David Koll <davidkoll@web.de>	test	Wed, 13 Jun 2007 23:03:57 +0200
Tim Waage <tim.waage@web.de>	Bachelorarbeitsauszug	Wed, 13 Jun 2007 23:11:03 +0200
ih mehl <ihmehl@hotmail.com>	attachtestsubject	Fri, 22 Jun 2007 19:24:39 +0300
ih mehl <ihmehl@hotmail.com>	no attach	Fri, 22 Jun 2007 20:25:42 +0300
David Koll <davidkoll@web.de>	attach die 2.	Fri, 22 Jun 2007 19:22:34 +0200
David Koll <davidkoll@web.de>	just another test	Fri, 22 Jun 2007 22:24:09 +0200
David Koll <davidkoll@web.de>	yo_näh	Thu, 28 Jun 2007 23:40:29 +0200
Christian Menke <cmenke@davion.de>	UTF-8 attacks! hähähähähäh! :P	Thu, 28 Jun 2007 23:46:59 +0200
Christian Menke <cmenke@davion.de>	iso-mail..... öh!	Fri, 29 Jun 2007 11:07:19 +0200

view reply forward delete message

Settings successfully loaded!

GUI - ComposingMailWindow



GUI - SettingsWindow



Settings...

POP3:

Server Name: Port:

User Name:

Password:

delete messages on server

SMTP:

Server Name: Port:

use SMTP Authentication

User Name: plain

Password: login

CRAM-MD5

Data Containers developed for ih-mehl [1 / 2]



- **Most important: Email class to represent single email:**
- Parses plain text emails as received from the mail servers.
- Provides getters and setters for important headers; body
- Can „turn itself into XML“ and parse from XML – for HDD storage.
- New since last meeting: Some handling of character encodings for example for German umlauts. Handles ISO-8559-x and UTF-8 sections embedded into headers (quoted-printable or base64). More details later on.
- Used in the Mailbox class, in the POP3 handler, in the SMTP handler, in the GUI when composing a new mail. In other words: everywhere!

Data Containers developed for ih-mehl [2 / 2]



- **Mailbox class as a container for Email objects:**
- Internal storage method: Vector<EMail>
- Can add, delete, return single EMail
- Can load a mailbox from and store itself mailbox to a specified XML file on the HDD.
- Makes use of the MailboxParser class for loading a mailbox from file.
 - > Let's have a look at the XML markup used, then
 - > at the MailboxParser class

Mailbox storage on HDD



- **Examination of existing XML formats for storing emails revealed they were often overly complicated and not really what we needed.**
- ih-mehl now has its own, custom format (which is bad!)
- The format is so simple that anyone could write a SAX handler for it within a quarter of an hour (which is good!)
- The XML tree begins with a <mailbox> element that may have a name attribute.
- Followed by as many <mail> sub-elements as there are mails in the box.
- Each <mail> element may have one <hdr_from>, <hdr_to>, <hdr_subject>, <hdr_msgid>, <hdr_date> and <body> sub-element containing the actual mail data.
- That's it!

Handling special characters in emails [1 / 2]



- **Initially, emails were to be transferred as 7 bit ASCII (American Standard for Information Interchange). This causes a lot of problems with characters not used in the english language and therefore not in ASCII-128, f.ex. ä, ö, ü, €, ...**
Different approaches to solve this problem:
- Send hex code of problematic character's value (called „quoted-printable“):
Wer hat die Mails **=?ISO-8859-15?Q?gel=F6scht=3F?=
=? wrapper and charset required; encoded character prefixed by '='.
- required to avoid unwanted, accidental decoding.
- encoded sections must be parsed, hex codes (marked red in the example) can then decoded according to specified charset. (Done by Java's CharsetDecoder.)**
- Transform problematic character(s) to base 64 number system by combining three bytes of input into four 6-bit blocks of output ($2^6 = 64$). These 6-bit blocks, padded with zeros, can then be safely transferred:
Achtung, UTF-8 ab **=?UTF-8?B?w6RoaGhoIGhpZXI=?=
- transform the whole section back to base 256,
- decode the whole section according to charset (done by Java's CharsetDecoder)**

Handling special characters in emails [2 / 2]



- **Special encodings in mail headers are always „embedded“ as shown in the examples on the previous page. In the mail body, there usually is no special embedding.**
Most often: Content-Type header specifies "text/plain; charset=XYZ" and the body uses all 8 bits in a byte – no base64, no quoted-printable hex codes.
- The POP3 handler stores the received bytes in a Java String. Java Strings (and chars) internally use a 16 bit format. This causes a re-encoding: from whatever charset the mail body had, interpreted as whatever charset is the operating system's default charset, into Java's 16 bit format.
Afterwards the original encoding is broken so badly that it can't be recovered.
- The solution is to not try any decoding in the mail body at all. If the mail body's charset and the operating system charset overlap, the Java String itself decodes everything without problems.