The

Modern Mail Handler

a Master's Thesis

by

Markus Schnalke

advised by

Prof. Dr. Franz Schweiggert
and Dr. Andreas Borchert

Ulm University

2012

How do we convince people that in programming simplicity and clarity – in short: what mathematicians call "elegance" – are not a dispensable luxury, but a crucial matter that decides between success and failure?

— Edsger W. Dijkstra —

CONTENTS

Preface ..... **vii**

1 Introduction **1**

1.1 MH – the Mail Handler 1

1.2 nmh 4

1.3 mmh 5

2 Discussion **9**

2.1 Streamlining 10

2.1.1 Mail Transfer Facilities 10

2.1.2 Non-MUA Tools 13

2.1.3 Displaying Messages 14

2.1.4 Configure Options 15

2.1.5 Command Line Switches 19

2.2 Modernizing 24

2.2.1 Code Relics 24

2.2.2 Attachments 26

2.2.3 Signing and Encrypting 31

2.2.4 Draft and Trash Folder 32

2.2.5 Modern Defaults 34

2.3 Styling 35

2.3.1 Code Style 36

2.3.2 Structural Rework 37

2.3.3 Profile Reading 41

2.3.4 Standard Libraries 43

2.3.5 User Data Locations 45

2.3.6 Modularization 45

3 Summary **49**

A Tools of mmh **51**

References **53**

PREFACE

I have discovered the mail client *nmh* in fall 2009. At that time I used *mutt*, as many advanced Unix users do. When I read about nmh, its concepts convinced me at once. The transition from mutt to nmh was similar to beginning with file management in the Unix shell when being used to the *midnight commander*, or like starting with vi when being used to modeless editors. Such a change is not trivial, but, in being convinced by the concepts and by having done similar transitions for file management and editing already, it was not too difficult. In contrast, setting up nmh to a convenient state became a tedious task that took several months. Once having nmh arranged this way, I enjoyed using it because of its conceptional elegance and its scripting capabilities. Nevertheless, it was still inconvenient for handling attachments, non-ASCII character encodings, and similar features of modern emailing. My setup demanded more and more additional configuration and helper scripts to have nmh behave the way I wanted; yet my expectations were rather common for modern emailing. As a computer scientist and programmer, I wanted to improve the situation.

In spring 2010, I sent a message [mail: mta-mua] to the *nmh-workers* mailing list [web: nmh-workers], asking for the possibility to offer a Google Summer of Code [web: gsoc] project for me. Participating in the development of nmh in this manner appeared attractive to me, because I would have been able to work full time on nmh. Although the nmh community had reacted generally positive to the suggestion, the administrative work for such a project would had been too much. Nonetheless, my proposal had activated the nmh community. In the following weeks, goals for nmh's future were discussed. In these discussions, I became involved in the question whether nmh should include mail transfer facilities [mail: mta-mua]. I argued for the Mail Transfer Agent of nmh to be removed. In this fundamental question, my opinion differed from the opinion of most others. Sadly, besides the discussions, hardly any real work was done. Being unable to work on nmh in a way that would be accepted at university as part of my studies, I needed to choose another project.

Half a year later, starting in August 2010, I took one semester off to travel through Latin America. During my time in Argentina, I wanted to work on free software. This brought me back to nmh. Richard Sandelman, an active nmh user, took care of the official basis. Juan Granda, an Argentine free software developer, organized a computer with Internet connection. Thanks to them, I was able to work on nmh during my three-month stay in Santiago del Estero, Argentina. Quickly it became obvious that I would not succeed with my main goal, to improve the character encoding handling. (One of its ramifications is the missing transfer decoding of quoted text in replies.) As this is one of the most intricate parts of the system, the goal was simply set too high. Instead, I improved the code base as I read through it. I found minor bugs for which I proposed fixes. Additionally, I improved the documentation in minor ways. When I started to work on larger code changes, I had to discover that the community was reluctant to change. Its wish for compatibility was much stronger than its wish for convenient out-of-the-box setups – in contrast to my opinion. Once again, this led to long discussions. I came to understand their point of view, but it is different from mine. At the end of my three-month project, I had become familiar with nmh's code base and community, I had improved the project in minor ways, and I still was convinced that I wanted to continue to do so.

Another half year later, the end of my studies came within reach. I needed to choose a topic for my master's thesis. Without question, I wanted to work on nmh. But not exactly on nmh, because I had accepted that its community has different goals than I have. Working on nmh would result in much discussion and, in consequence, little progress. After careful thought, I decided to start an experimental version of nmh. I wanted to implement my own ideas of how an MH-like system should look like. I wanted to create a usable alternative version to be compared with the present state of nmh. Eventually, my work would be proven successful or not. In any case, the nmh project would profit from my experiences.

Focus of this Document

This document explains the design goals and implementation decisions for mmh, an experimental version of nmh. It discusses technical, historical, social and philosophical considerations. On the technical side, this document explains how an existing project was streamlined by removing rough edges and better exploitation of the central concepts. On the historical side, changes through time are discussed, regarding the use cases and the email features, as well as the reactions to them. Socially, this document describes the effects and experiences of a newcomer with revolutionary aims entering an old and matured software project. Philosophical thoughts on style, mainly based on the Unix philosophy, are present throughout the discussions. The document describes the changes to nmh, but as well, it clarifies my personal perception of the concepts of MH and Unix, and explain my therefrom resulting point of view.

This document is written for the community around MH-like mail systems, including developers and users. Despite the focus on MH-like systems, this document may be valuable to anyone interested in the Unix philosophy and anyone in contact with old software projects, be it code- or community-related.

The reader is expected to be familiar with Unix, C and emailing. Good Unix shell knowledge is required, because MH relies fundamentally on the shell. Without the power of the shell, MH becomes a motorcycle without winding roads: boring. Introductions to Unix and its shell can be found in *The UNIX Programming Environment* by Kernighan and Pike [Kernighan84] or *The UNIX System* by Bourne [Bourne83]. The reader is assumed to be a C programmer, but the document should be understandable otherwise, too. The definitive guide to C is Kernighan and Ritchie's *The C Programming Language* [Kernighan88]. A book about system-level C programming, such as those written by Rochkind and Curry [Rochkind85, Curry96], can be helpful as additional literature. Old books are likely more helpful for understanding, because large parts of the source code are old. The reader is expected to know the format of email messages and the structure of email transfer systems, at least on a basic level. It's advisable to have cross-read RFC 821 and RFC 822. Furthermore, basic understanding of MIME [RFC 2045–2049] is good to have. The Wikipedia provides good introduction-level information about email [web: email].

Frequent references to the Unix philosophy will be made. Gancarz has tried to sum it up in his book *The UNIX Philosophy* [Gancarz95]. Even better, though less concrete, are *The UNIX Programming Environment* [Kernighan84] and *The Practice of Programming* [Kernighan99] by Kernighan and Pike. The term paper *Why the Unix Philosophy still matters* [Schnalke10] by myself provides an overview on the philosophy, including a case study of MH.

Although a brief introduction to MH is provided in Section 1.1, the reader is encouraged to have a look at *MH & nmh: Email for Users & Programmers* by Jerry Peek [Peek95]. The current version is available freely on the Internet. It is the definitive guide to MH and nmh.

This document is neither a user's tutorial to mmh nor an introduction to any of the topics covered. The technical discussions are on an advanced level. Nevertheless, as knowledge of the fundamental concepts is the most valuable information a user can acquire about some program or software system, this document may be worth a read for non-developers as well.

Organization

This thesis consists of three chapters. Chapter 1 introduces into the topic, describing MH and explaining the background and goals of the mmh project. Chapter 2 discusses the work done in the project. It is organized along the three major goals of the project, namely streamlining, modernizing, and styling. Not every change is described because that would bore the reader. Instead, important changes and those standing for a set of similar changes are described and discussed. Chapter 3 finishes up by summarizing the achievements and taking a look into the future of the mmh project.

Italic font is used for names of software projects, literature, and man pages, as well as to emphasize new terms. `Constant width` font is used to denote names of programs, files, functions, command lines, code excerpts, program input and output, and the like.

References to man pages are printed as "*cat*(1)", which is a reference to the man page of `cat` in section one of the Unix manual. *Requests for Comments* (RFCs), which describe the working of the Internet, are referenced as "RFC 821". Literature is cited in brackets, such as "[Kernighan84]". Citations of email messages and websites are distinguished by "mail:" and "web:" prefixes. All references are collected at the end of the document. Links to the software projects mentioned throughout the text are collected there as well.

This document describes practical programming work. The code of mmh is managed with the `git` version control system. All code changes were checked in. In the discussions, references to corresponding code changes are printed as "[☞ `1a2b3c4`]". The identifier is the seven-letter-prefix of the changeset's hash value, which is considered unique. Any change can be looked up in the repository, using the command line '`git show 1234567`'. ('`1234567`' needs to be replaced with the concrete hash value.) At the time of writing, changesets could be looked up online at: `http://git.marmaro.de/?p=mmh;a=commitdiff;h=1234567`. But as we all know, URIs are always at risk to change.

Acknowledgments

Prof. Dr. Schweiggert and Dr. Borchert, you made my work possible with your believe in the relevance of this topic. I enjoyed my time studying in Ulm in the environment around you. For me, this had been just the right place to be. You have enriched my studies greatly and I have learned a lot from you both. Thank you.

This thesis is my own work, but it would be in much poorer shape without the help of friends. They spotted errors, bad explanations, weak argumentations, and clumsy sentences. Their comments and suggestions shaped this document; they made it focused and comprehensive.

The lady comes first, but not for that reason alone.

Katja Rogers
: Your help concerning the English language and writing in general was outstanding. The reader will surely recognize the sentences I have never showed to you.

Dominik Lang
: You were mmh's first user and you were convinced of it at once. That was great! Thanks for your feedback and suggestions regarding mmh, as well as for improving this document.

Michael Müller
: Thanks for your corrections, comments, and for the discussions on typography.

Matthias Matousek
: Thanks for your corrections, comments, and for sharing your ''living room''.

Boris Kraut
: Thanks for your corrections, comments, and for the idea to use an adjusted mmh setup as an address book.

Aaron Müller
: You have surprised me by proposing yourself to being a proof reader. Thanks for your valuable annotations and quick responses.

Marius Glöggler and Martina Schnalke
: You have covered the tedious tasks of double-checking the changeset references and the bibliography, respectively. Thank you.

My family supported me in times when I was busy; Jean-Luc Picard and his crew took me on diverting space adventures in my lazy times.

Yet, no one of the aforementioned persons is connected to my work as close as Lydi is. — All those days, already in Argentina, when I was working long hours and even on holidays instead of spending time with you. All the explanations of always the same doubts and difficulties I suffered from, and you just listened. All the delays near the end of my work though you have wished for an earlier finish. In spite of it all, you knew what it meant to me.

Danke für dein Verständnis.

Chapter 1
INTRODUCTION

MH is a set of mail handling tools with a common concept. It is similar to the Unix tool chest, which is a set of file handling tools with a common concept. *nmh* is the currently most popular implementation of an MH-like mail handling system. This thesis describes an experimental version of nmh, named *mmh*.

This chapter introduces MH, its history, concepts and how it is used. It describes nmh's code base and community to give the reader a better understanding of the project's condition at the time when mmh started off. Furthermore, this chapter outlines the mmh project itself, describing the motivation for it and its goals.

1.1 MH – THE MAIL HANDLER

MH is a conceptual email system design and its concrete implementation. MH had started as a design proposal at RAND Corporation, where the first implementation followed later. In spirit, MH is similar to Unix, which influenced the world more in being a set of system design concepts than in being a specific software product. The ideas behind Unix are summarized in the *Unix philosophy* [Gancarz95]. MH follows this philosophy.

History

In 1977 at RAND Corporation, Norman Shapiro and Stockton Gaines proposed the design of a new mail handling system [MH-Memo], to superseed RAND's old monolithic *Mail System* (MS). One year later, in 1978, Bruce Borden picked up on the proposal and implemented a prototype, which he called *Mail Handler* (MH). Before the prototype's existence, the concept was believed to be practically unusable. But the prototype – written in only three weeks – proved successful and replaced MS thereafter. [Anderson89, p. 4]

In the early eighties, the University of California at Irvine (UCI) started to use MH. Marshall T. Rose and John L. Romine then became the driving force. They took over the development and pushed MH forward [Anderson89, p. 4]. RAND had put the code into the public domain by then. MH was developed at UCI at the same time when the Internet appeared, BSD started to support TCP/IP networking, and Eric Allman wrote Sendmail. MH was extended as emailing became more featured. The development of MH was closely related to the development of email RFCs. In the advent of the *Multipurpose Internet Mail Extensions* (MIME), MH was one of the first implementations of the new email standard.

In the nineties, the Internet became popular and in December 1996, Richard Coleman initiated the *New Mail Handler* (nmh) project. Nmh is a fork of MH 6.8.3 and bases heavily on the *LBL changes* by Van Jacobson, Mike Karels and Craig Leres [web: lbl]. Colman intended to modernize MH and improve its portability and MIME handling capabilities. The development of MH at UCI stopped after the 6.8.4 release in February 1996, soon after the development of nmh had started. Today, nmh is developed openly in the Internet community. It has almost completely replaced the original MH. Some systems might still provide the old MH, but hardly for good reasons.

In the last years, the majority of changes in nmh was maintenance work. Nevertheless, the development was revived in December 2011 and stayed busy since then.

Concepts

MH consists of a set of tools, each covering a specific task of email handling, such as composing a message, replying to a message, refiling a message to a different folder, listing the messages in a folder. The tools are invoked directly from the Unix shell [Anderson89].

The tools operate on a common mail storage, which consists of *mail folders* (directories) and *messages* (regular files). Each message is stored in a separate file [Anderson89]. The files are named with ascending numbers in each folder. The specific format of the mail storage characterizes MH in the same way as the format of the file system characterizes Unix.

MH tools maintain a *context*, which includes for instance the current mail folder. Processes in Unix have a similar context, containing the current working directory, for instance. In contrast, the process context is maintained by the Unix kernel automatically, whereas MH tools need to maintain the MH context themselves. The user can have one MH context or multiple ones; he can even share it with others.

Messages are named by their numeric filename, but they can have symbolic names, as well. These are either one of six system-controlled position names and a shorthand for the range of all messages, or user-settable group names for arbitrary sets of messages. These names are called sequences. Automatically updated position names exist for the first, last, previous, next, current message, and for the number one beyond the last message. (In mmh, the names of these sequences are abbreviated to the first character.) User-defined sequences can be bound to the folder containing the messages (*public sequences*) or to the user's context (*private sequences*).

The user's *profile* is the file that contains his MH configuration. Default switches for the individual tools can be specified to adjust them to the user's personal preferences. These switches will be automatically supplied whenever the specific tool is invoked. Additionally, a single command can be linked under different names with different default values. Form templates for new messages and replies, as well as format files to adjust the output of tools are easily exchanged in the profile. Almost every part of the system can be adjusted to personal preference.

The whole system is well scriptable and extensible. New MH tools are built out of or on top of existing ones quickly. MH encourages the user to tailor, extend, and automate the system. As the MH tool chest was modeled after the Unix tool chest, the properties of the latter apply to the former as well.

Using MH

Many tutorials to using MH [Rose86, Moss88, Hegardt90] are old, but still they teach the concepts and basics, which remained unchanged. Rose and Romine have written an excellent introduction on a more technical level, with pointers to advanced usage [Rose85]. For a more recent document, it is strongly recommended to have a look at the *MH Book* [Peek95, Part II], especially at its online version.

Following here is a sample mail handling session with mmh. Details might vary to MH and nmh but the look and feel is the same.

```
$ inc
Incorporating new mail into inbox...

   1+ 2012-07-04 23:42  Bob                 The Unix philosophy
   2  2365-05-15 02:17  "Jean-Luc Picard"   Good advice

$ show                 (display the current message, i.e. the one marked with '+')
Date:    Wed, 04 Jul 2012 23:42:00 +0200
From:    Bob <bob@example.org>
To:      meillo@marmaro.de
Subject: The Unix philosophy

part       text/plain                 307
The design of MH follows the Unix philosophy.

> At the heart of the Unix philosophy is the idea that
> the power of a system comes more from the relationship
> among programs than from the programs themselves.

        -- Brian W. Kernighan and Rob Pike
                ``The UNIX Programming Environment´´
                        (quotation freely rearranged)

$ refile +quotes       (move the current message to the folder 'quotes')
Create folder "/home/meillo/Mail/quotes"? y

$ next                 (display the next message; equal to 'show n')
Date:    Sat, 15 May 2365 02:17:00 +0000
From:    "Jean-Luc Picard" <captain@uss-enterprise>
To:      meillo@marmaro.de
Subject: Good advice

part       text/plain                  89
Open your mind to the past -- art, history, philosophy.
And all this may mean something.
```

```
$ repl                  (reply to this message)
[...]                   (the reply is composed in an editor session)

What now? send

$ folder                (print information on the current folder)
inbox+ has 1 message   (2-2); cur=2

$ scan                  (list the messages in the current folder)
   2+ 2365-05-15 02:17  "Jean-Luc Picard"  Good advice

$ scan +quotes          (list the messages in folder 'quotes')
   1  2012-07-04 23:42  Bob                The Unix philosophy

$
```

1.2 NMH

In order to understand the condition, goals and dynamics of a project, one needs to know the reasons behind them. This section gives background information.

MH predates the Internet; it comes from times before networking was universal; it comes from times when emailing was small, short and simple. Then, MH grew, spread and adapted to the changes email went through. Its core concepts, however, remained the same. During the eighties, students at UCI actively worked on MH. They added new features and optimized the code for the systems popular at that time. This was in times before POSIX and ANSI C. As large parts of the code stem from this time, today's nmh source code still contains many ancient parts. BSD-specific code and constructs tailored for hardware of that time are frequent.

Nmh started about one decade after the POSIX and ANSI C standards were released. A more modern coding style entered the code base but still a part of the developers were "of the old type". The developer base became more diverse, thus broadening the range of different coding styles. Programming practices from different decades merged in the project. As several peers added code, the system became more a conglomeration of single tools rather than a homogeneous of-one-cast mail system. For that, leadership would have been necessary. Nevertheless, MH's basic concepts held the project together. They were mostly untouched throughout the years.

Though clearly separated on the surface – as a collection of small, separate programs – the source code turns out to be fairly interwoven. Multiple separate components are compiled into a program for efficiency reasons. This leads to intricate innards.

It is visible in nmh that the advent of MIME raised the complexity of email by a magnitude. The MIME-related parts are the most complex ones. It is also visible that MIME support was added on top of the old MH core. MH's tool chest style made this easily possible and encourages such approaches, but unfortunately, it led to duplicated functions and half-hearted implementation of concepts.

To provide backward-compatibility, it is a common understanding in the nmh community to not change the default settings. In consequence, users need to activate modern features explicitly to be able to use them. The ancient style in which fresh nmh setups remain to appear causes difficulties for new users, as modern email features require additional configuration. The small but mature community around nmh, however, needs little change as they have had their convenient setups for decades.

1.3 MMH

I started to work on my experimental version in October 2011, basing my work on nmh version *nmh-1.3-dev*. At that time no more than three commits were made to nmh since the beginning of 2011, the latest one being [☞ a01a41d], committed on 2011-04-13. In December, when I announced my work in progress on the nmh-workers mailing list [mail: mmh-ann], nmh's community became active, all of a sudden. This movement was heavily pushed by Paul Vixie's ''edginess'' comment [mail: edginess]. After long years of stagnation, nmh became actively developed again. Hence, while I was working on mmh, the community was working on nmh, in parallel but unrelated.

The name *mmh* may stand for *modern mail handler*, because the project tries to modernize nmh. Personally however, I prefer to call mmh *meillo's mail handler*, emphasizing that the project is my version of nmh, following my visions and preferences. (My login name is *meillo*.) This project model was inspired by *dwm*, which is Anselm Garbe's personal window manager – targeted to satisfy Garbe's personal needs whenever conflicts appear. Dwm has retained its lean elegance and its focused character [web: sloc-dwm], whereas its community-driven predecessor *wmii* had grown fat over time [web: sloc-wmii]. The development of mmh should remain focused.

Motivation

MH is the most important of very few email systems in a tool chest style. Tool chests are powerful because they can be perfectly automated and extended. They allow the implementation of arbitrary kinds of front-ends on top of the tool chest quickly and without internal knowledge. Additionally, tool chests are easier to maintain than monolithic programs [Gancarz95, p. 14]. MH-like email tool chests should be kept alive as they fill a market niche by providing conceptional elegance and unique scripting qualities. Mmh tries to create a modern and convenient entry point to MH-like systems for new and interested users.

The mmh project is motivated by deficits of nmh and by my wish for general changes. At the time the mmh project started, nmh had not yet adjusted to modern emailing needs well enough. The default setup was completely unusable for modern emailing. Too much setup work was required. Several modern features were already available, but the community did not want to have them active by default. Mmh is my way to change this.

In my eyes, MH's concepts could be exploited better and the style of the tools could be improved. Both would simplify and generalize the system, providing cleaner interfaces and greater software leverage at the same time. Mmh is my way to demonstrate this.

In providing multiple parts of the email system, nmh can hardly compete with the large specialized projects that focus on one of the components only. The situation could be improved by concentrating the development power on the most unique part of MH and letting the user pick his preferred set of other mail components. Today's pre-packaged software components encourage this model. Mmh is my way to provide this.

It is worthwhile to fork nmh for the development of mmh, because the two projects focus on different goals and differ in fundamental questions. The nmh community's reluctance regarding change conflicts with my strong desire for it [mail: nmh-goal]. In developing a separate experimental version, new approaches can easily be tried out without the need to discuss changes beforehand. In fact, revolutionary changes are hardly possible otherwise.

The mmh project provides the basis on which the aforementioned ideas can be implemented and demonstrated, without the need to change the nmh project or its community. Of course, the results of the mmh project shall improve nmh, in the end. By no means it is my intent to work against the nmh project.

Target Field

Any effort needs to be targeted towards a specific goal in order to be successful. Therefore, a description of an imagined typical mmh user follows. Actually, as mmh is my personal version of MH, this is sort of a description of myself. Developing software for one's own is a reliable way to produce software that matches the user's desires.

The target user of mmh likes Unix and its philosophy. He appreciates to use programs that are conceptionally appealing. He is familiar with the command line and enjoys its power. He is capable of shell scripting and wants to improve his productivity by scripting the mail system. He uses modern email features, such as attachments, non-ASCII text, digital signatures and message encryption in a natural way. He is able to set up mail system components and likes to pick the ones he prefers. He has a reasonably modern operating system that complies to the POSIX and ANSI C standards.

The typical user invokes mmh commands directly in an interactive shell session, even on workstations where graphical front-ends could be added. Likely, he runs his mail setup on a server machine, to which he connects via ssh. He might automate mail processing with mmh tools but definitely he uses the tools to build better tools. In any case, he wants to have the flexibility to change his setup to suit his needs.

The typical mmh user is a programmer. He likes to, occasionally, make use of the opportunity of free software by putting hands on and getting involved in software he uses. In consequence, he likes small and clean code bases and cares for code quality. In general, he believes that:

- The elegance of source code is most important.
- Concepts are more important than concrete implementations.
- Code optimizations for anything but readability should be avoided.
- Removed code is debugged code.
- Having a lot of choice is bad.

Goals of the mmh Project

Streamlining
: Mmh should be stripped down to its core, which is the user agent (MUA). The feature set should be distilled to the indispensable ones, effectively removing corner cases. Parts that do not add to the main task of being a conceptionally appealing user agent should be removed. This includes the mail transfer and mail retrieval facilities. Choice should be reduced to the main options. All tools should be tightly shaped.

Modernizing
: Mmh's feature set needs to become more modern. Better support for attachments, digital signatures, and message encryption should be added. MIME support should be integrated deeper and more naturally. The modern email features need to be readily available, out-of-the-box. On the other hand, obsolete facilities can be dropped out and ancient technologies need not be further supported. The available concepts should be expanded as far as possible. A small set of concepts should recur consistently.

Styling
: Mmh's source code should be updated to modern standards. Standardized library functions should replace non-standard versions whenever possible. Code should be separated into distinct modules when feasible. Time and space optimizations should to be replaced by clear and readable code. A uniform programming style should prevail. The whole system should appear to be of-one-style; it should feel like being cast as one.

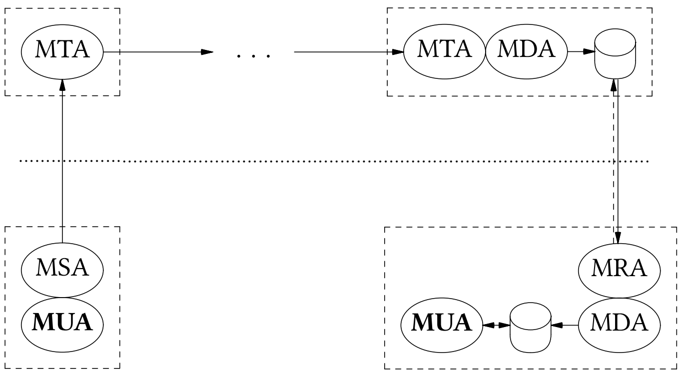
Chapter 2
DISCUSSION

This main chapter discusses the practical work accomplished in the mmh project. It is structured along the goals chosen for the project. A selection of the work undertaken is described.

This discussion compares the present version of mmh with the state of nmh at the time when the mmh project had started, i.e. fall 2011. Recent changes in nmh are rarely part of the discussion.

Whenever lines of code are counted, David A. Wheeler's *sloccount* was used to measure the amount in a comparable way.

For the reader's convenience, the structure of modern email systems is depicted in the following figure. It illustrates the path a message takes from sender to recipient.



The ellipses denote mail agents, i.e. different jobs in email processing. These are:

Mail User Agent (MUA)
: The only program users directly interact with. It includes functions to compose new mail, display received mail, and to manage the mail storage. It is called a *mail client* as well.

Mail Submission Agent (MSA)
: A special kind of Mail Transfer Agent, used to submit mail into the mail transport system. Often it is also called an MTA.

Mail Transfer Agent (MTA)
: A node in the mail transport system. It transfers incoming mail to a transport node nearer to the final destination. An MTA may be the final destination itself.

Mail Delivery Agent (MDA)
: Delivers mail according to a set of rules. Usually, the messages are stored to disk.

Mail Retrieval Agent (MRA)
: Initiates the transfer of mail from a remote location to the local machine. (The dashed arrow in the figure represents the pull request.)

The dashed boxes represent entities that usually reside on single machines. The box on the lower left represents the sender's system. The box on the upper left represents the first mail transfer node. The box on the upper right represents the transfer node responsible for the destination address. The box on the lower right represents the recipient's system. Often, the boxes above the dotted line are servers on the Internet. Many mail clients, including nmh, include all of the components below the dotted line. This is not the case for mmh; it implements the MUA only.

2.1 STREAMLINING

MH once provided a complete email system. The community around nmh tries to keep nmh in similar shape. In fundamental contrast, mmh shall be an MUA only. I believe that the development of all-in-one mail systems is obsolete. Today, email is too complex to be fully covered by a single project. Such a project will not be able to excel in all aspects. Instead, the aspects of email should be covered by multiple projects, which then can be combined to form a complete system. Excellent implementations for the various aspects of email already exist. Just to name three examples: Postfix is a specialized MTA, Procmail is a specialized MDA, and Fetchmail is a specialized MRA. I believe that it is best to use such specialized tools instead of providing the same function once more as a side component.

Doing something well requires focusing on a small set of specific aspects. Under the assumption that development which is focussed on a particular area produces better results there, specialized projects will be superior in their field of focus. Hence, all-in-one mail system projects – no matter if monolithic or modular – will never be the best choice in any of the fields. Even in providing the most consistent all-in-one system, they are likely to be beaten by projects that focus exclusively on the creation of a homogeneous system by integrating existing mail components.

Usually, the limiting resource in the community development of free software is man power. If the development effort is spread over a large development area, it becomes more difficult to compete with the specialists in the various fields. The concrete situation for MH-based mail systems is even tougher, given their small and aged community, concerning both developers and users.

In consequence, I believe that the available development resources should focus on the point where MH is most unique. This is clearly the user interface – the MUA. Peripheral parts should be removed to streamline mmh for the MUA task.

2.1.1 Mail Transfer Facilities

The removal of the mail transfer facilities, effectively dropping the MSA and MRA, had been the first work task in the mmh project. The desire for this change initiated the creation of the mmh project.

Focusing on one mail agent role only, is motivated by Eric Allman's experience with Sendmail. He identified the limitation of Sendmail to the MTA task as one reason for its success [Costales08, p. xviii]:

> Second, I limited myself to the routing function – I wouldn't write user agents or delivery back-ends. This was a departure of the dominant thought of the time, in which routing logic, local delivery, and often the network code were incorporated directly into the user agents.

In nmh, the MSA is called *Message Transfer Service* (MTS). This facility, implemented by the `post` command, establishes network connections and spoke SMTP to submit messages to be relayed to the outside world. When email transfer changed, this part needed to be changed as well. Encryption and authentication for network connections needed to be supported, hence TLS and SASL were introduced into nmh. This added complexity without improving the core functions. Furthermore, keeping up with recent developments in the field of mail transfer requires development power and specialists. In mmh, this whole facility was simply cut off [☞ f6aa95b] [☞ fecd5d3] [☞ 156d35f]. Instead, mmh depends on an external MSA. All outgoing mail in mmh goes through the `sendmail` command, which almost any MSA provides. If not, a wrapper program can be written. It must read the message from the standard input, extract the recipient addresses from the message header, and hand the message over to the MSA. For example, a wrapper script for qmail would be:

```
#!/bin/sh
exec qmail-inject  # ignore command line arguments
```

The requirement to parse the recipient addresses out of the message header may be removed in the future. Mmh could pass the recipient addresses as command line arguments. This appears to be the better interface.

To retrieve mail, the `inc` command in nmh acts as MRA. It establishes network connections and speaks POP3 to retrieve mail from remote servers. As with mail submission, the network connections required encryption and authentication, thus TLS and SASL were added to nmh. Support for message retrieval through IMAP will soon become necessary additions, too, and likewise for any other changes in mail transfer. But not in mmh because it has dropped the support for retrieving mail from remote locations [☞ ab7b484]. Instead, it depends on an external tool to cover this task. Mmh has two paths for messages to enter mmh's mail storage: (1) Mail can be incorporated with `inc` from the system maildrop, or (2) with `rcvstore` by reading them, one at a time, from the standard input.

With the removal of the MSA and MRA, mmh converted from a complete mail system to only an MUA. Now, of course, mmh depends on third-party software. An external MSA is required to transfer mail to the outside world; an external MRA is required to retrieve mail from remote machines. Excellent implementations of such software exist. They likely are superior to the internal versions that were removed. Additionally, the best suiting programs can be chosen freely.

As it had already been possible to use an external MSA and MRA, why should the internal version not be kept for convenience? Transferred to a different area, the question, whether there is sense in having a fall-back pager in all the command line tools for the cases when `more` or `less` are not available, appears to be ridiculous. Of course, MSAs and MRAs are more complex than text pagers and not necessarily available but still the concept of orthogonal design holds: “Write programs that do one thing and do it well” [Salus94, McIlroy78]. Here, this part of the Unix philosophy was applied not only to the programs but to the project itself. In other words: Develop projects that focus on one thing and do it well. Projects which have grown complex should be split, for the same reasons that programs which have grown complex should be split. If it is conceptionally more elegant to have the MSA and MRA as separate projects then they should be separated. In my opinion, this is the case. The RFCs suggest this separation by clearly distinguishing the different mail handling tasks [RFC 821]. The small interfaces between the mail agents support the separation as well.

Once, email had been small and simple. At that time, `/bin/mail` had covered everything there was to email and still was small and simple. Later, the essential complexity of email increased. (Essential complexity is the complexity defined by the problem itself [Brooks86].) Consequently, email systems grew. RFCs started to introduce the concept of mail agents to separate the various roles because they became more extensive and because new roles appeared. As mail system implementations grew, parts of them were split off. For instance, a POP server was included in the original MH; it was removed in nmh. Now is the time to go one step further and split off the MSA and MRA, as well. Not only does this decrease the code size of the project, more importantly, it unburdens mmh of the whole field of message transfer, with all its implications for the project. There is no more need for concern with changes in network transfer. This independence is gained by depending on external components that cover the field.

In general, functionality can be added in three different ways:

1. By implementing the function in the project itself.
2. By depending on a library that provides the function.
3. By depending on a program that provides the function.

While implementing the function in the project itself leads to the largest increase in code size and requires the most maintenance and development work, it keeps the project’s dependence on other software lowest. Using libraries or external programs requires less maintenance work but introduces dependencies on external projects. Programs have the smallest interfaces and provide the best separation, but possibly limit the information exchange. External libraries are more strongly connected than external programs, thus information can be exchanged in a more flexible manner. Obviously, adding code to a project increases the maintenance work. As implementing complex functions in the project itself adds a lot of code, this should be avoided if possible. Thus, the dependencies only change in their character, not in their existence. In mmh, library dependencies on `libsasl2` and `libcrypto`/`libssl` were traded against program dependencies on an MSA and an MRA. This also meant trading build-time dependencies against run-time dependencies. Besides providing stronger separation and greater flexibility, program dependencies also allowed over 6 000 lines of code to be removed from mmh. This made mmh’s code base about 12 % smaller. Reducing the project’s code size by such an amount without actually losing functionality is a convincing argument. Actually, as external MSAs and MRAs are likely superior to the project’s internal versions, the common user even gains functionality.

Users of MH should not have problems setting up an external MSA and MRA. Also, the popular MSAs and MRAs have large communities and a lot of available documentation. Choices for MSAs range from small forwarders such as *ssmtp* and *nullmailer*, over mid-size MTAs including *masqmail* and *dma*, up to full-featured MTAs as for instance *Postfix*. MRAs are provided for example by *fetchmail*, *getmail*, *mpop*, and *fdm*.

2.1.2 Non-MUA Tools

One goal of mmh is to remove the tools that do not significantly contribute to the MUA's job. Loosely related and rarely used tools distract from a lean appearance, and require maintenance work without adding much to the core task. By removing these tools, mmh became more streamlined and focused.

- `conflict` was removed [☞ 8b23509] because it is a mail system maintenance tool and not MUA-related. It even checked `/etc/passwd` and `/etc/group` for consistency, which is completely unrelated to email. A tool like `conflict` is surely useful, but it should not be shipped with mmh.
- `rcvtty` was removed [☞ 14767c9] because its use case of writing to the user's terminal on reception of mail is obsolete. If users like to be informed of new mail, the shell's `MAILPATH` variable or graphical notifications are technically more appealing. Writing to terminals directly is hardly ever desired today. If, though, one prefers this approach, the standard tool `write` can be used in a way similar to:

```
scan -file - | write `id -un`
```

- `viamail` was removed [☞ eda72d6] when the new attachment system was activated, because `forw` could then cover the task itself. The `sendfiles` shell script was rewritten as a wrapper around `forw` [☞ 0e82199].
- `msgchk` was removed [☞ bb9360e], because it lost its use case when POP support was removed. A call to `msgchk` provided hardly more information than:

```
ls -l /var/mail/meillo
```

  Yet, it distinguished between old and new mail, but these details can be retrieved with `stat`(1), too. A small shell script could be written to print the information in a similar way, if truly necessary. As mmh's `inc` only incorporates mail from the user's local maildrop, and thus no data transfers over slow networks are involved, there is hardly any need to check for new mail before incorporating it.
- `msh` was removed [☞ 9166901] because the tool was in conflict with the philosophy of MH. It provided an interactive shell to access the features of MH. However, it was not just a shell tailored to the needs of mail handling, but one large program that had several MH tools built in. This conflicted with the major feature of MH of being a tool chest. `msh`'s main use case had been accessing Bulletin Boards, which have ceased to be popular.

Removing msh together with the truly archaic code relics vmh and wmh saved more than 7 000 lines of C code – about 15 % of the project's original source code amount. Having less code – with equal readability, of course – for the same functionality is an advantage. Less code means less bugs and less maintenance work. As rcvtty and msgchk are assumed to be rarely used and can be implemented in different ways, why should one keep them? Removing them streamlined mmh. viamail's use case is now partly obsolete and partly covered by forw, hence there is no reason to still maintain it. conflict is not related to the mail client, and msh conflicts with the basic concept of MH. These two tools might still be useful, but they should not be part of mmh.

Finally, there is slocal, which is an MDA and thus not directly MUA-related. It should be removed from mmh because including it conflicts with the idea that mmh is an MUA only. However, slocal provides rule-based processing of messages, like filing them into different folders, which is otherwise not available in mmh. Although slocal neither pulls in dependencies, nor does it include a separate technical area (cf. Sec. 2.1.1), it still accounts for about 1 000 lines of code that need to be maintained. As slocal is almost self-standing, it should be split off into a separate project. This would cut the strong connection between the MUA mmh and the MDA slocal. For anyone not using MH, slocal would become yet another independent MDA, like *procmail*. Then slocal could be installed without a complete MH system. Likewise, mmh users could decide to use *procmail* without having a second, unused MDA, i.e. slocal, installed. That appears to be conceptionally the best solution. Yet, slocal is not split off. I defer the decision over slocal out of a need for deeper investigation. In the meanwhile, it remains part of mmh as its continued existence is not significant; slocal is unrelated to the rest of the project.

2.1.3 Displaying Messages

Since the very beginning, already in the first concept paper [MH-Memo], show had been MH's message display program. show mapped message numbers and sequences to files and invoked mhl to have the files formatted. With MIME, this approach was not sufficient anymore. MIME messages can consist of multiple parts. Some parts, like binary attachments or text content in foreign charsets, are not directly displayable. show's understanding of messages and mhl's display capabilities could not cope with the task any longer.

Instead of extending these tools, additional tools were written from scratch and were added to the MH tool chest. Doing so is encouraged by the tool chest approach. Modular design is a great advantage for extending a system, as new tools can be added without interfering with existing ones. First, the new MIME features were added in form of the single program mhn. The command 'mhn -show 42' had then shown the message number 42, interpreting MIME. With the 1.0 release of nmh in February 1999, Richard Coleman finished the split of mhn into a set of specialized tools, which together covered the multiple aspects of MIME. One of them was mhshow, which replaced 'mhn -show'. It was capable of displaying MIME messages appropriately.

From then on, two message display tools were part of nmh, show and mhshow. To ease the life of users, show was extended to automatically hand the job over to mhshow if displaying the message would be beyond show's abilities. In consequence, the user would simply invoke show (possibly through next or prev) and get the message printed with either show or mhshow, whatever was more appropriate.

Having two similar tools for basically the same task is redundancy. Usually, users do not distinguish between `show` and `mhshow` in their daily mail reading. Having two separate display programs was therefore unnecessary from a user's point of view. Besides, the development of both programs needed to be in sync, to ensure that the programs behaved in a similar way, because they were used like a single tool. Different behavior would have surprised the user.

Today, non-MIME messages are rather seen to be a special case of MIME messages, although it is the other way round. As `mhshow` already had been able to display non-MIME messages, it appeared natural to drop `show` in favor of using `mhshow` exclusively [☞ 4c1efdd]. Removing `show` is no loss in function, because `mhshow` covers it completely. Yet, the old behavior of `show` can still be emulated with the simple command line:

```
mhl `mhpath c`
```

For convenience, `mhshow` was renamed to `show` after `show` was gone. It is clear that such a rename may confuse future developers when trying to understand the history. Nevertheless, I consider the convenience on the user's side, to outweigh the inconvenience for understanding the evolution of the tools.

To prepare for the transition, `mhshow` was reworked to behave more like `show` first (cf. Sec. 2.1.3). Once the tools behaved more alike, the replacing appeared to be even more natural. Today, mmh's new `show` has become the one single message display program once again, with the difference that today it handles MIME messages as well as non-MIME messages. The outcomes of the transition are one program less to maintain, no second display program for users to deal with, and less system complexity.

Still, removing the old `show` hurts in one regard: It had been such a simple program. Its lean elegance is missing from the new `show`, but there is no alternative; supporting MIME demands higher essential complexity.

2.1.4 Configure Options

Customization is a double-edged sword. It allows better suiting setups, but not for free. There is the cost of code complexity to be able to customize. There is the cost of less tested setups, because there are more possible setups and especially corner cases. Steve Johnson confirms [Raymond04, p. 233]:

> Unless it is done very carefully, the addition of an on/off configuration option can lead to a need to double the amount of testing. Since in practice one never does double the amount of testing, the practical effect is to reduce the amount of testing that any given configuration receives. Ten options leads to 1024 times as much testing, and pretty soon you are talking real reliability problems.

Additionally, there is the cost of choice itself. The code complexity directly affects the developers. Less tested code affects both users and developers. The problem of choice affects the users, for once by having to choose but also by more complex interfaces that require more documentation. Whenever options add few advantages but increase the complexity of the system, they should be considered for removal. I have reduced the number of project-specific configure options from 15 to 3.

Mail Transfer Facilities

With the removal of the mail transfer facilities (cf. Sec. 2.1.1) five configure options vanished:

The switches `--with-tls` and `--with-cyrus-sasl` had activated the support for transfer encryption and authentication. They are not needed anymore. [☞ fecd5d3] [☞ 156d35f]

The configure switch `--enable-pop` had activated the message retrieval facility. Whereas the code area that had been conditionally compiled in for TLS and SASL support was small, the conditionally compiled code area for POP support was much larger. The code base had only changed slightly on toggling TLS or SASL support but it had changed much on toggling POP support. The changes in the code base could hardly be overviewed. By having POP support togglable, a second code base had been created, one that needed to be tested. This situation is basically similar for the conditional TLS and SASL code, but there the changes are minor and can yet be overviewed. Still, conditional compilation of a code base creates variations of the original program. More variations require more testing and maintenance work.

Two other options had only specified default configuration values: `--with-mts` defined the default transport service [☞ f6aa95b]. With `--with-smtpservers` default SMTP servers could be set [☞ 128545e]. Both of them became irrelevant when the SMTP transport service was removed. In mmh, all messages are handed over to `sendmail` for transportation.

Backup Prefix

The backup prefix is the string that was prepended to message filenames to tag them as deleted. By default it had been the comma character (‘`,`’). In July 2000, Kimmo Suominen introduced the configure option `--with-hash-backup` to change the default to the hash character ‘`#`’. This choice was probably personal preference, but, being related or not, words that start with the hash character introduce a comment in the Unix shell. Thus, the command line ‘`rm #13 #15`’ calls `rm` without arguments because the first hash character starts a comment that reaches until the end of the line. To delete the backup files, ‘`rm ./#13 ./#15`’ needs to be used. Thus, using the hash as backup prefix may be seen as a precaution against backup loss.

First, I removed the configure option but added the profile entry `Backup-Prefix`, which allowed to specify an arbitrary string as backup prefix [☞ 6c40d48]. This change did not remove the choice but moved it to a location where it suited better, in my eyes.

Eventually however, the new trash folder concept (cf. Sec. 2.2.4) removed the need for the backup prefix completely. [☞ 8edc5aa] [☞ ca0b3e8]

Editor and Pager

The two configure options `--with-editor=EDITOR` `--with-pager=PAGER` were used to specify the default editor and pager at configure time. Doing so at configure time made sense in the eighties, when the set of available editors and pagers varied much across different systems. Today, the situation is more homogeneous. The programs `vi` and `more` can be expected to be available on every Unix system, as they are specified by

POSIX since two decades. (The specifications for `vi` and `more` appeared in [XVS87] and [XCU92], respectively.) As a first step, these two tools were hard-coded as defaults [☞ `5d43a99`]. Not changed were the `editor` and `moreproc` profile entries, which allowed the user to override the system defaults. Later, the concept was reworked again to respect the standard environment variables `VISUAL` and `PAGER` if they are set. Today, mmh determines the editor to use in the following order, taking the first available and non-empty item [☞ `f85f4b7`]:

1. Environment variable `MMHEDITOR`
2. Profile entry `Editor`
3. Environment variable `VISUAL`
4. Environment variable `EDITOR`
5. Command `vi`.

The pager to use is determined in a similar order [☞ `0c4214e`]:

1. Environment variable `MMHPAGER`
2. Profile entry `Pager` (replaces `moreproc`)
3. Environment variable `PAGER`
4. Command `more`.

By respecting the `VISUAL`/`EDITOR` and `PAGER` environment variables, the new behavior complies with the common style on Unix systems. It is more uniform and clearer for users.

ndbm

`slocal` used to depend on the database library *ndbm*. The database is used to store the `Message-ID:` header field values of all messages delivered. This enabled `slocal` to suppress delivering the same message to the same user twice. This features was enabled by the `-suppressdup` switch.

Because a variety of versions of the database library exist [Wolter04], complicated autoconf code was needed to detect them correctly. Furthermore, the configure switches `--with-ndbm=ARG` and `--with-ndbmheader=ARG` were added to help with difficult setups that would not be detected automatically or not correctly.

By removing the suppress duplicates feature of `slocal`, the dependency on *ndbm* vanished and 120 lines of complex autoconf code could be saved [☞ `ecd6d6a`]. The change removed functionality but that is considered minor to the improvement of dropping the dependency and the complex autoconf code.

MH-E Support

The configure option `--disable-mhe` was removed when the MH-E support was reworked. MH-E is the Emacs front-end to MH. It requires MH to provide minor additional functions. The `--disable-mhe` configure option had switched off these extensions. After removing the support for old versions of MH-E, only the `-build` switches of `forw` and `repl` are left to be MH-E extensions. They are now always built in because they add little code and complexity. In consequence, the `--disable-mhe` configure option was removed [☞ `a7ce7b4`]. Dropping the option also removed a variant of the code base that would have needed to be tested. This change was undertaken in January 2012 in nmh and thereafter merged into mmh.

Masquerading

The configure option `--enable-masquerade` could take up to three arguments: `draft_from`, `mmailid`, and `username_extension`. They activated different types of address masquerading. All of them were implemented in the SMTP-speaking `post` command. Address masquerading is an MTA's task and mmh does not cover this field anymore. Hence, true masquerading needs to be implemented in the external MTA.

The *mmailid* masquerading type is the oldest one of the three and the only one available in the original MH. It provided a *username* to *fakeusername* mapping, based on the `passwd`'s GECOS field. Nmh's man page *mh-tailor*(5) described the use case as being the following:

> This is useful if you want the messages you send to always appear to come from the name of an MTA alias rather than your actual account name. For instance, many organizations set up 'First.Last' sendmail aliases for all users. If this is the case, the GECOS field for each user should look like: ''First [Middle] Last <First.Last>''

As mmh sends outgoing mail via the local MTA only, the best location to do such global rewrites is there. Besides, the MTA is conceptionally the right location because it does the reverse mapping for incoming mail (aliasing), too. Furthermore, masquerading set up there is readily available for all mail software on the system. Hence, mmailid masquerading was removed. [☞ `0836c80`]

The *username_extension* masquerading type did not replace the username but would append a suffix, specified by the `USERNAME_EXTENSION` environment variable, to it. This provided support for the *user-extension* feature of qmail [Sill02, p. 141] and the similar *plussed user* processing of Sendmail [Costales08, p. 476]. The decision to remove this username_extension masquerading was motivated by the fact that `spost` had not supported it yet. Username extensions can be used in mmh, but less convenient. [☞ `2abae0b`]

The *draft_from* masquerading type instructed `post` to use the value of the `From:` header field as SMTP envelope sender. Sender addresses could be replaced completely. Mmh offers a kind of masquerading similar in effect, but with technical differences. As mmh does not transfer messages itself, the local MTA has final control over the sender's address. Any masquerading mmh introduces may be reverted by the MTA. In times of pedantic spam checking, an MTA will take care to use sensible envelope sender addresses to keep its own reputation up. Nonetheless, the MUA can set the `From:` header field and thereby propose a sender address to the MTA. The MTA may then decide to take that one or generate the canonical sender address for use as envelope sender address. [☞ `b14ea60`]

In mmh, the MTA will always extract the recipient and sender from the message header (`sendmail`'s `-t` switch). The `From:` header field of the draft may be set arbitrary by the user. If it is missing, the canonical sender address will be generated by the MTA.

Remaining Options

Two configure options remain in mmh. One of them is the file locking method to use: `--with-locking=[dot|fcntl|flock|lockf]`. The idea of removing all methods except the portable *dot locking* and having that one as the default is appealing, but this change requires deeper technical investigation into the topic. The other remaining option, `--enable-debug`, compiles the programs with debugging symbols. This option is likely to stay.

2.1.5 Command Line Switches

The command line switches of MH tools are similar in style to the switches in the X Window System. They consist of a single dash ('`-`') followed by a word. For example '`-truncate`'. To ease typing, the switch can be abbreviated, given the remaining prefix is unambiguous. If no other switch starts with the letter 't', then any of '`-truncate`', '`-trunc`', '`-tr`', and '`-t`' is equal. As a result, switches can neither be grouped (as in '`ls -ltr`') nor can switch arguments be appended directly to the switch itself (as in '`sendmail -q30m`'). Many switches have negating counter-parts, which start with 'no'. For example '`-notruncate`' inverts the '`-truncate`' switch. They exist to override the effect of default switches in the profile. Every program in mmh has two generic switches: `-help`, to print a short message on how to use the program, and `-Version` (with capital 'V'), to tell what version of mmh the program belongs to.

Switches change the behavior of programs. Programs that do one thing in one way require no switches. In most cases, doing something in exactly one way is too limiting. If one task should be accomplished in various ways, switches are a good approach to alter the behavior of a program. Changing the behavior of programs provides flexibility and customization to users, but at the same time it complicates the code, the documentation, and the usage of the program. Therefore, the number of switches should be kept small. A small set of well-chosen switches is best. Usually, the number of switches increases over time. Already in 1985, Rose and Romine have identified this as a major problem of MH [Rose85, p. 12]:

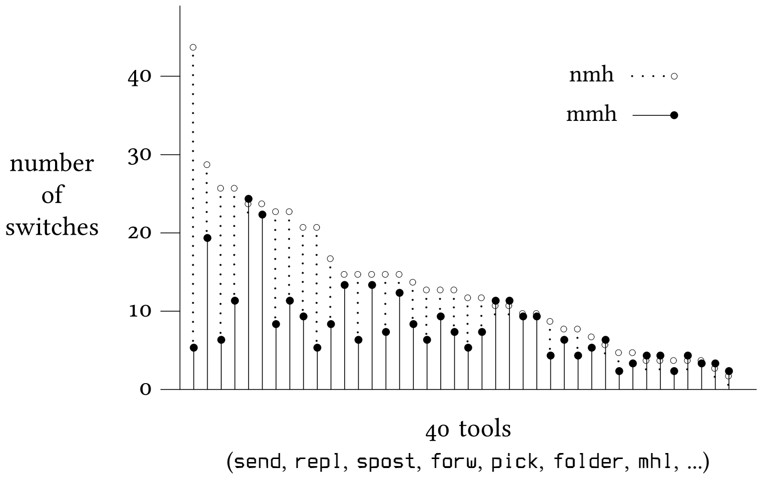
> A complaint often heard about systems which undergo substantial development by many people over a number of years, is that more and more options are introduced which add little to the functionality but greatly increase the amount of information a user needs to know in order to get useful work done. This is usually referred to as creeping featurism.
>
> Unfortunately MH, having undergone six years of off-and-on development by ten or so well-meaning programmers (the present authors included), suffers mightily from this.

Being reluctant to adding new switches (or *options*, as Rose and Romine call them) is one part of a counter-action, the other part is removing hardly used switches. Nmh's tools have lots of switches already implemented. Hence, cleaning up by removing some of them was the more important part of the counter-action. Removing existing functionality is always difficult because it breaks programs that use these functions. Also, for every obsolete feature, there'll always be someone who still uses it and thus opposes its removal. This puts the developer into the position, where sensible improvements to style are regarded as destructive acts. Yet, living with the featurism is far worse, in my eyes, because future needs will demand adding further features, worsening the situation more and more. Rose and Romine added in a footnote, "[...] send will no doubt acquire an endless number of switches in the years to come" [Rose85, p. 12]. Although clearly humorous, the comment points to the nature of the problem. Refusing to add any new switches would encounter the problem at its root, but this is not practical. New needs will require new switches and it would be unwise to block them strictly. Nevertheless, removing obsolete switches still is an effective approach to deal with the problem. Working on an experimental branch without an established user base, eased my work because I did not offend users when I removed existing functions.

Rose and Romine counted 24 visible and 9 more hidden switches for `send`. In nmh, they increased up to 32 visible and 12 hidden ones. At the time of writing, no more than 4 visible switches and 1 hidden switch have remained in mmh's `send`. These numbers include the two generic switches, `-help` and `-Version`.

Hidden switches are ones not documented. In mmh, 12 tools have hidden switches. 9 of them are `-debug` switches, the other 6 provide special interfaces for internal use.

The following figure displays the number of switches for each of the tools that is available in both nmh and mmh. The tools are sorted by the number of switches they had in nmh. Both visible and hidden switches were counted, but not the generic help and version switches. Whereas in the beginning of the project, the average tool had 11 switches, now it has no more than 5 – only half as many. If the 'no' switches and similar inverse variant are folded onto their counter-parts, the average tool had 8 switches in pre-mmh times and has 4 now. The total number of functional switches in mmh dropped from 465 to 233.



A part of the switches vanished after functions were removed. This was the case for network mail transfer, for instance. Sometimes, however, the work flow was the other way: I looked through the *mh-chart*(7) man page to identify the tools with apparently too many switches. Then I valued the benefit of each switch by examining the tool's man page and source code, aided by literature research and testing.

Draft Folder Facility

A change early in the project was the complete transition from the single draft message to the draft folder facility [☞ 337338b] (cf. Sec. 2.2.4). The draft folder facility was introduced in the mid-eighties, when Rose and Romine called it a "relatively new feature" [Rose85]. Since then, the facility was included, inactive by default. By making it permanently active and by related rework of the tools, the `-[no]draftfolder`, and `-draftmessage` switches could be removed from `comp`, `repl`, `forw`, `dist`, `whatnow`, and `send` [☞ 337338b]. The only flexibility lost with this change is having multiple draft folders within one profile. I consider this a theoretical problem only. At the same time, the `-draft` switch of `anno`, `refile`, and `send` was removed. The special treatment of *the* draft message became irrelevant after the rework of the draft system (cf. Sec. 2.2.4).

In Place Editing

`anno` had the switches `-[no]inplace` to either annotate the message in place and thus preserve hard links, or annotate a copy to replace the original message. The latter approach broke hard links. Following the assumption that linked messages should truly be the same message and annotating it should not break the link, the `-[no]inplace` switches were removed and the previous default `-inplace` was made the definitive behavior [☞ c8195b4]. The `-[no]inplace` switches of `repl`, `forw`, and `dist` could be removed, as well, as they were simply passed through to `anno`.

`burst` also had `-[no]inplace` switches, but with a different meaning. With `-inplace`, the digest had been replaced by the table of contents (i.e. the introduction text) and the burst messages were placed right after this message, renumbering all following messages. Also, any trailing text of the digest was lost, though, in practice, it usually consists of an end-of-digest marker only. Nonetheless, this behavior appeared less elegant than the `-noinplace` behavior, which already had been the default. Nmh's *burst*(1) man page reads:

> If `-noinplace` is given, each digest is preserved, no table of contents is produced, and the messages contained within the digest are placed at the end of the folder. Other messages are not tampered with in any way.

The decision to drop the `-inplace` behavior was supported by the code complexity and the possible data loss it caused. `-noinplace` was chosen to be the definitive behavior. [☞ 68a686a]

Forms and Format Strings

Historically, the tools that had `-form` switches to supply a form file had `-format` switches as well to supply the contents of a form file as a string on the command line directly. In consequence, the following two lines equaled:

```
scan -form scan.mailx
scan -format "`cat /path/to/scan.mailx`"
```

The `-format` switches were dropped in favor for extending the `-form` switches [☞ f51956b]. If their argument starts with an equal sign ('='), then the rest of the argument is taken as a format string, otherwise the arguments is treated as the name of a format file. Thus, now the following two lines equal:

```
scan -form scan.mailx
scan -form "=`cat /path/to/scan.mailx`"
```

This rework removed the prefix collision between `-form` and `-format`. Typing '`-fo`' is sufficient to specify form file or format string.

The different meaning of `-format` for `forw` and `repl` was removed in mmh. `forw` was completely switched to MIME-type forwarding, thus removing the `-[no]format` [☞ `6e27160`]. For `repl`, the `-[no]format` switches were reworked to `-[no]filter` switches [☞ `67411b1`]. The `-format` switches of `send` and `post`, which had a third meaning, were removed likewise [☞ `f3cb7cd`]. Eventually, the ambiguity of the `-format` switches is resolved by not having such switches anymore in mmh.

MIME Tools

The MIME tools, which once were part of `mhn` (whatever that stood for), had several switches that added little practical value to the programs. The `-[no]realsize` switches of `mhbuild` and `mhlist` were removed [☞ `8d8f1c3`]. Real size calculations are done always now because nmh's *mhbuild*(1) man page states that "This provides an accurate count at the expense of a small delay" with the small delay not being noticeable on modern systems.

The `-[no]check` switches were removed together with the support for `Content-MD5:` header fields [RFC 1864] (cf. Sec. 2.2.1) [☞ `31dc797`].

The `-[no]ebcdicsafe` and `-[no]rfc934mode` switches of `mhbuild` were removed because they are considered obsolete [☞ `01a3480`] [☞ `3363e26`].

Content caching of external MIME parts, activated with the `-rcache` and `-wcache` switches was completely removed [☞ `d1fefd9`]. External MIME parts are rare today, having a caching facility for them appears to be unnecessary.

In pre-MIME times, `mhl` had covered many tasks that are part of MIME handling today. Therefore, `mhl` could be simplified to a large extend, reducing the number of its switches from 21 to 6 [☞ `350ad6d`] [☞ `0e46503`].

Header Printing

`folder`'s data output is self-explaining enough that displaying the header line makes little sense. Hence, the `-[no]header` switch was removed and headers are never printed [☞ `601cc73`].

In `mhlist`, the `-[no]header` switches were removed, as well [☞ `b24f965`]. In this case, the headers are printed always because the output is not self-explaining.

`scan` also had `-[no]header` switches. Printing this header had been sensible until the introduction of format strings made it impossible to display column headings. Only the folder name and the current date remained to be printed. As this information can be perfectly generated with `folder` and `date`, the switches were removed [☞ `c477dc5`].

By removing all `-header` switches, the collision with `-help` on the first two letters was resolved. Currently, `-h` evaluates to `-help` for all tools of mmh.

Suppressing Edits or the Invocation of the WhatNow Shell

The `-noedit` switch of `comp`, `repl`, `forw`, `dist`, and `whatnow` was removed and replaced by the ability to specify `-editor` with an empty argument [☞ 75fca31]. (Using '`-editor /bin/true`' is nearly the same. It differs only in setting the previous editor.)

The more important change is the removal of the `-nowhatnowproc` switch [☞ ee4f43c]. This switch had once introduced an awkward behavior, as explained in nmh's man page for *comp*(1):

> The `-editor` *`editor`* switch indicates the editor to use for the initial edit. Upon exiting from the editor, `comp` will invoke the `whatnow` program. See *whatnow*(1) for a discussion of available options. The invocation of this program can be inhibited by using the `-nowhatnowproc` switch. (In truth of fact, it is the `whatnow` program which starts the initial edit. Hence, `-nowhatnowproc` will prevent any edit from occurring.)

Effectively, the `-nowhatnowproc` switch caused only only a draft message to be created. As '`-whatnowproc /bin/true`' does the same, the `-nowhatnowproc` switch was removed for being redundant.

Various

- With the removal of MMDF maildrop format support, `packf` and `rcvpack` no longer needed the `-mbox` and `-mmdf` switches. The behavior of `-mbox` became the definitive behavior [☞ 3916ab6]. Further rework in both tools made the `-file` switch unnecessary [☞ ca10237].
- Mmh's tools do no longer clear the screen (`scan`'s and `mhl`'s `-[no]clear` switches [☞ e57b173] [☞ 943765e]). The message formating tool `mhl` does neither ring the bell (`-[no]bell` [☞ e11983f]) nor does it page the output itself (`-length` [☞ 5b9d883]) anymore. Generally, the pager to use is no longer specified with the `-[no]moreproc` command line switches for `mhl` and `show`/`mhshow` [☞ 39e87a7].
- In order to avoid prefix collisions among switch names, the `-version` switch was renamed to `-Version` (with capital 'V') [☞ 32b2354]. Every program has the `-version` switch but its first three letters collided with the `-verbose` switch, present in many programs. The rename solved this problem once for all. Although this rename breaks a basic interface, having the `-V` abbreviation to display the version information, isn't all too bad.
- `-[no]preserve` of `refile` was removed [☞ 8edc5aa] because what use was it anyway? Quoting nmh's man page *refile*(1):

  > Normally when a message is refiled, for each destination folder it is assigned the number which is one above the current highest message number in that folder. Use of the `-preserv` [sic!] switch will override this message renaming, and try to preserve the number of the message. If a conflict for a particular folder occurs when using the `-preserve` switch, then `refile` will use the next available message number which is above the message number you wish to preserve.

- The removal of the `-[no]reverse` switches of `scan` [☞ `8edc5aa`] is a bug fix. This is supported by the comments ‘‘–[no]reverse under #ifdef BERK (I really HATE this)’’ by Rose and ‘‘Lists messages in reverse order with the ‘–reverse’ switch. This should be considered a bug’’ by Romine in the changelogs. The question remains why neither Rose nor Romine have fixed this bug in the eighties when they wrote these comments.

2.2 MODERNIZING

In the more than thirty years of MH’s existence, its code base was increasingly extended. New features entered the project and became alternatives to the existing behavior. Relics from several decades have gathered in the code base but seldom obsolete features were dropped. This section describes the removing of old code and the modernizing of the default setup. It focuses on the functional aspect only; the non-functional aspects of code style are discussed in Sec. 2.3.1.

2.2.1 Code Relics

My position regarding the removal of obsolete code is much more revolutional than the nmh community appreciates. Working on an experimental version, I was able to quickly drop functionality that I considered ancient. The need for consensus with peers would have slowed this process down. Without the need to justify my decisions, I was able to rush forward.

In December 2011, Paul Vixie motivated the nmh developers to just do the work [mail: edginess]:

> let’s stop walking on egg shells with this code base. there’s no need to discuss whether to keep using vfork, just note in [sic!] passing, [...] we don’t need a separate branch for removing vmh or ridding ourselves of #ifdef’s or removing posix replacement functions or depending on pure ansi/posix ‘‘libc’’.
>
> these things should each be a day or two of work and the ‘‘main branch’’ should just be modern. [...] let’s push forward, aggressively.

I did so already in the months before. I pushed forward. I simply dropped the cruft.

The decision to drop a feature was based on literature research and careful thinking, but whether having had contact with this particular feature within my own computer life served as a rule of thumb. I explained my reasons in the commit messages in the version control system. Hence, others can comprehend my view and argue for undoing the change if I have missed an important aspect. I was quick in dropping parts. I rather include falsely dropped parts again, than going at a slower pace. Mmh is experimental work; it requires tough decisions.

Process Forking

Being a tool chest, MH creates many processes. In earlier times `fork()` had been an expensive system call, because the process’s image needed to be completely duplicated at once. This expensive work was especially unnecessary in the commonly occurring case wherein the image is replaced by a call to `exec()` right after having forked the child process. The `vfork()` system call was invented to speed up this particular case. It completely omits the duplication of the image. On old systems this resulted in significant speed ups. Therefore MH used `vfork()` whenever possible.

Modern memory management units support copy-on-write semantics, which make `fork()` almost as fast as `vfork()`. The man page of *vfork*(2) in FreeBSD 8.0 states:

> This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of vfork() as it will, in that case, be made synonymous to fork(2).

Vixie supports the removal with the note that "the last system on which fork was so slow that an mh user would notice it, was Eunice. that was 1987" [mail: edginess]. I replaced all calls to `vfork()` with calls to `fork()` [☞ `40821f5`].

Related to the costs of `fork()` is the probability of its success. In the eighties, on heavy loaded systems, calls to `fork()` were prone to failure. Hence, many of the `fork()` calls in the code were wrapped into loops to retry the `fork()` several times, to increase the chances to succeed eventually. On modern systems, a failing `fork()` call is unusual. Hence, in the rare case when `fork()` fails, mmh programs simply abort [☞ `5fbf37e`].

Header Fields

- The `Encrypted:` header field was introduced by RFC 822, but already marked as legacy in RFC 2822. Today, OpenPGP provides the basis for standardized exchange of encrypted messages [RFC 4880, RFC 3156]. Hence, the support for `Encrypted:` header fields is removed in mmh [☞ `064527f`].
- The native support for `Face:` header fields has been removed, as well [☞ `8e5be81`]. This feature is similar to the `X-Face:` header field in its intent, but takes a different approach to store the image. Instead of encoding the image data directly into the header field, it contains the hostname and UDP port where the image date can be retrieved. There is even a third Face system, which is the successor of `X-Face:`, although it re-uses the `Face:` header field name. It was invented in 2005 and supports colored PNG images. None of the Face systems described here is popular today. Hence, mmh has no direct support for them.
- The `Content-MD5:` header field was introduced by RFC 1864. It provides detection of data corruption during the transfer. But it can not ensure verbatim end-to-end delivery of the contents [RFC 1864]. The proper approach to verify content integrity in an end-to-end relationship is the use of digital signatures [RFC 4880]. On the other hand, transfer protocols should detect corruption during the transmission. The TCP includes a checksum field therefore. These two approaches in combinations render the `Content-MD5:` header field superfluous. Not a single one out of 4 200 messages from two decades in the nmh-workers mailing list archive [web: nmh-workers] contained a `Content-MD5:` header field. Neither did any of the 60 000 messages in my personal mail storage. Removing the support for this header field [☞ `31dc797`], removed the last place where MD5 computation was needed. Hence, the MD5 code could be removed as well. Over 500 lines of code vanished by this one change.

MMDF maildrop support

This type of maildrop format is conceptionally similar to the mbox format, but uses a different message delimiter ('\1\1\1\1', commonly written as '^A^A^A^A', instead of 'From '). Mbox is the de-facto standard maildrop format on Unix, whereas the MMDF maildrop format is now forgotten. Mbox remains as the only packed mailbox format, supported in mmh.

The simplifications within the code were moderate. Mainly, the reading and writing of MMDF mailbox files was removed. But also, switches of packf and rcvpack could be removed [☞ 3916ab6]. In the message parsing function sbr/m_getfld.c, knowledge of MMDF packed mail boxes was removed [☞ 684ec30]. Further code structure simplifications may be possible there, because only one single packed mailbox format is left to be supported. I have not worked on them yet because m_getfld() is heavily optimized and thus dangerous to touch. The risk of damaging the intricate workings of the optimized code is too high.

Prompter's Control Keys

The program prompter queries the user to fill in a message form. When used as 'comp -editor prompter', the resulting behavior is similar to mailx. Apparently, prompter had not been touched lately. Otherwise it's hardly explainable why it still offered the switches -erase *chr* and -kill *chr* to name the characters for command line editing. The times when this had been necessary are long time gone. Today these things work out-of-the-box, and if not, are configured with the standard tool stty. The switches are removed now [☞ 0bd9750].

Hardcopy Terminal Support

More of a funny anecdote is a check for being connected to a hardcopy terminal. It remained in the code until spring 2012, when I finally removed it [☞ b7764c4]. The check only prevented a pager to be placed between the printing program (mhl) and the terminal. In nmh, this could have been ensured statically with the -nomoreproc at the command line, too. In mmh, setting the profile entry Pager or the environment variable PAGER to cat is sufficient.

2.2.2 Attachments

The mind model of email attachments is unrelated to MIME. Although the MIME RFCs [RFC 2045–2049] define the technical requirements for having attachments, they do not mention the term. Instead of attachments, MIME talks about "multi-part message bodies" [RFC 2045], a more general concept. Multi-part messages are messages "in which one or more different sets of data are combined in a single body" [RFC 2046]. MIME keeps its descriptions generic; it does not imply specific usage models. Today, one usage model is prevalent: attachments. The idea is having a main text document with files of arbitrary kind attached to it. In MIME terms, this is a multi-part message having a text part first and parts of arbitrary type following.

MH's MIME support is a direct implementation of the RFCs. The perception of the topic described in the RFCs is clearly visible in MH's implementation. As a result, MH had all the MIME features but no idea of attachments. But users do not need all the MIME features, they want convenient attachment handling.

Composing MIME Messages

In order to improve the situation on the message composing side, Jon Steinhart had added an attachment system to nmh in 2002 [☞ 7480dbc]. In the file docs/README-ATTACHMENTS, he described his motivation to do so:

> Although nmh contains the necessary functionality for MIME message handing [sic!], the interface to this functionality is pretty obtuse. There's no way that I'm ever going to convince my partner to write mhbuild composition files!

With this change, the mind model of attachments entered nmh. In the same document:

> These changes simplify the task of managing attachments on draft files. They allow attachments to be added, listed, and deleted. MIME messages are automatically created when drafts with attachments are sent.

Unfortunately, the attachment system, like every new facilities in nmh, was inactive by default.

During my time in Argentina, I tried to improve the attachment system. But, after long discussions my patch died as a proposal on the mailing list because of great opposition in the nmh community [mail: attach]. In January 2012, I extended the patch and applied it to mmh [☞ 8ff284f]. In mmh, the attachment system is active by default. Instead of command line switches, the Attachment-Header profile entry is used to specify the name of the attachment header field. It is pre-defined to Attach:.

To add an attachment to a draft, a header line needs to be added:

```
To: bob
Subject: The file you wanted
Attach: /path/to/the/file-bob-wanted
--------
Here it is.
```

The header field can be added to the draft manually in the editor, or by using the 'attach' command at the WhatNow prompt, or non-interactively with anno:

```
anno -append -nodate -component Attach -text /path/to/attachment
```

Drafts with attachment headers are converted to MIME automatically by send. The conversion to MIME is invisible to the user. The draft stored in the draft folder is always in source form with attachment headers. If the MIMEification fails (e.g. because the file to attach is not accessible) the original draft is not changed.

The attachment system handles the forwarding of messages, too. If the attachment header value starts with a plus character ('+'), like in 'Attach: +bob 30 42', the given messages in the specified folder will be attached. This allowed to simplify forw [☞ f41f04c].

Closely related to attachments is non-ASCII text content, because it requires MIME as well. In nmh, the user needed to call 'mime' at the WhatNow prompt to have the draft converted to MIME. This was necessary whenever the draft contained non-ASCII characters. If the user did not call 'mime', a broken message would be sent. Therefore, the automimeproc profile entry could be specified to have the 'mime' command invoked automatically each time. Unfortunately, this approach conflicted with the attachment system because the draft would already be in MIME format at the time when the attachment system wanted to MIMEify it. To use nmh's attachment system, 'mime' must not be called at the WhatNow prompt and `automimeproc` must not be set in the profile. But then the case of non-ASCII text without attachment headers was not caught. All in all, the solution was complex and irritating. My patch from December 2010 [mail: attach] would have simplified the situation.

Mmh's current solution is even more elaborate. Any necessary MIMEification is done automatically. There is no 'mime' command at the WhatNow prompt anymore. The draft will be converted automatically to MIME when either an attachment header or non-ASCII text is present. Furthermore, the hash character ('`#`') is not special any more at line beginnings in the draft message. Users need not concern themselves with the whole topic at all. The approach taken in mmh is tailored towards today's most common case: a text part, possibly with attachments. This case was simplified.

Although the new approach does not anymore support arbitrary MIME compositions directly, the full power of `mhbuild` can still be accessed. Given no attachment headers are included, users can create `mhbuild` composition drafts like in nmh. Then, at the WhatNow prompt, they can invoke '`edit mhbuild`' to convert the draft to MIME. Because the resulting draft neither contains non-ASCII characters nor has it attachment headers, the attachment system will not touch it.

MIME Type Guessing

From the programmer's point of view, the use of `mhbuild` composition drafts had one notable advantage over attachment headers: The user provides the appropriate MIME types for files to include. The new attachment system needs to find out the correct MIME type itself. This is a difficult task. Determining the correct MIME type of content is partly mechanical, partly intelligent work. Forcing the user to find out the correct MIME type, forces him to do partly mechanical work. Letting the computer do the work can lead to bad choices for difficult content. For mmh, the latter option was chosen to spare the user the work [☞ `3baec23`].

Determining the MIME type by the suffix of the file name is a dumb approach, yet it is simple to implement and provides good results for the common cases. If no MIME type can be determined, text content is sent as 'text/plain', anything else under the generic fall-back type 'application/octet-stream'. Mmh implements this approach in the `print-mimetype` script [☞ `4b59442`].

A far better, though less portable, approach is the use of `file`. This standard tool tries to determine the type of files. Unfortunately, its capabilities and accuracy varies from system to system. Additionally, its output was only intended for human beings, but not to be used by programs. Nevertheless, modern versions of GNU `file`, which are prevalent on the popular GNU/Linux systems, provide MIME type output in machine-readable form. Although this solution is system-dependent, it solves the difficult problem well. On systems where GNU `file`, version 5.04 or higher, is available it should be used. One needs to specify the following profile entry to do so:

```
Mime-Type-Query: file -b --mime
```

Other versions of `file` might possibly be usable with wrapper scripts that reformat the output. The diversity among `file` implementations is great; one needs to check the local variant.

It is not possible in mmh to override the automatic MIME type guessing for a specific file. To do so, either the user would need to know in advance for which file the automatic guessing fails or the system would require interaction. I consider both cases impractical. The existing solution should be sufficient. If not, the user may always fall back to `mhbuild` composition drafts and bypass the attachment system.

Storing Attachments

Extracting MIME parts of a message and storing them to disk is performed by `mhstore`. The program has two operation modes, `-auto` and `-noauto`. With the former one, each part is stored under the filename given in the MIME part's meta information, if available. This naming information is usually available for modern attachments. If no filename is available, this MIME part is stored as if `-noauto` would have been specified. In the `-noauto` mode, the parts are processed according to the rules that are defined by `mhstore-store-*` profile entries. These rules define generic filename templates for storing or commands to post-process the contents in arbitrary ways. If no matching rule is available the part is stored under a generic filename, built from message number, MIME part number, and MIME type.

The `-noauto` mode had been the default in nmh because it was considered safe, in contrast to the `-auto` mode. In mmh, `-auto` is not dangerous anymore. Two changes were necessary:

1. Any directory path is removed from the proposed filename. Thus, the files are always stored in the expected directory. [☞ `41b6ead`]
2. Tar files are not extracted automatically any more. Thus, the rest of the file system will not be touched. [☞ `94c8004`]

In mmh, the result of '`mhstore -auto`' can be foreseen from the output of '`mhlist -verbose`'. Although the `-noauto` mode is considered to be more powerful, it is less convenient and `-auto` is safe now. Additionally, storing attachments under their original name is intuitive. Hence, `-auto` serves better as the default option [☞ `3410b68`].

Files are stored into the directory given by the `Nmh-Storage` profile entry, if set, or into the current working directory, otherwise. Storing to different directories is only possible with `mhstore-store-*` profile entries.

Still existing files get overwritten silently in both modes. This can be considered a bug. Yet, each other behavior has its draw-backs, too. Refusing to replace files requires adding a `-force` switch. Users will likely need to invoke `mhstore` a second time with `-force`. Eventually, only the user can decide in the specific case. This requires interaction, which I like to avoid if possible. Appending a unique suffix to the filename is another bad option. For now, the behavior remains as it is.

In mmh, only MIME parts of type message are special in `mhstore`'s `-auto` mode. Instead of storing message/rfc822 parts as files to disk, they are stored as messages into the current mail folder. The same applies to message/partial, although the parts are automatically reassembled beforehand. MIME parts of type message/external-body are not automatically retrieved anymore. Instead, information on how to retrieve them is output. Not supporting this rare case saved nearly one thousand lines of code [☞ 55e1d8c]. The MIME type 'application/octet-stream; type=tar' is not special anymore. The automatically extracting of such MIME parts had been the dangerous part of the `-auto` mode [☞ 94c8004].

Showing MIME Messages

The program `mhshow` was written to display MIME messages. It implemented the conceptional view of the MIME RFCs. Nmh's `mhshow` handles each MIME part independently, presenting them separately to the user. This does not match today's understanding of email attachments, where displaying a message is seen to be a single, integrated operation. Today, email messages are expected to consist of a main text part plus possibly attachments. They are no more seen to be arbitrary MIME hierarchies with information on how to display the individual parts. I adjusted `mhshow`'s behavior to the modern view on the topic.

One should note that this section completely ignores the original `show` program, because it was not capable to display MIME messages and is no longer part of mmh (cf. Sec. 2.1.3). Although `mhshow` was renamed to `show` in mmh, this section uses the name `mhshow`, in order to avoid confusion.

In mmh, the basic idea is that `mhshow` should display a message in one single pager session. Therefore, `mhshow` invokes a pager session for all its output, whenever it prints to a terminal [☞ a4197ea]. In consequence, `mhl` does no more invoke a pager [☞ 0e46503]. With `mhshow` replacing the original `show`, the output of `mhl` no longer goes to the terminal directly, but through `mhshow`. Hence, `mhl` does not need to invoke a pager. The one and only job of `mhl` is to format messages or parts of them. The only place in mmh, where a pager is invoked is `mhshow`.

Only text content is displayed. Other kinds of attachments are ignored. Non-text content needs to be converted to text by appropriate `mhshow-show-*` profile entries before, if this is possible and wanted. A common example for this are PDF files.

MIME parts are always displayed serially. The request to display the MIME type 'multipart/parallel' in parallel is ignored. It is simply treated as 'multipart/mixed' [☞ d0581ba]. This was already possible to request with the `-serialonly` switch of `mhshow`, which is now removed. As MIME parts are always processed exclusively, i.e. serially, the '`%e`' escape in `mhshow-show-*` profile entries became useless and was thus removed [☞ a20d405]. For parallel display, the attachments need to be stored to disk first.

To display text content in foreign charsets, they need to be converted to the native charset. Therefore, `mhshow-charset-*` profile entries were needed. In mmh, the conversion is performed automatically by piping the text through the `iconv` command, if necessary [☞ 2433122]. Custom `mhshow-show-*` rules for textual content might need a '`iconv -f %c %f |`' prefix to have the text converted to the native charset.

Although the conversion of foreign charsets to the native one has improved, it is not consistent enough. Further work needs to be done and the basic concepts in this field need to be re-thought. Though, the default setup of mmh displays message in foreign charsets correctly without the need to configure anything.

2.2.3 Signing and Encrypting

Nmh offers no direct support for digital signatures and message encryption. This functionality needed to be added through third-party software. In mmh, the functionality is included because it is a part of modern email and is likely wanted by users of mmh. A fresh mmh installation supports signing and encrypting out-of-the-box. Therefore, Neil Rickert's `mhsign` and `mhpgp` scripts [web: rickert] were included [☞ `f45cdc9`] [☞ `58cf09a`]. The scripts fit well because they are lightweight and similar of style to the existing tools. Additionally, no licensing difficulties appeared as they are part of the public domain.

`mhsign` handles the signing and encrypting part. It comprises about 250 lines of shell code and interfaces between `gnupg` and the MH system. It was meant to be invoked manually at the WhatNow prompt, but in mmh, `send` invokes `mhsign` automatically [☞ `c7b5e1d`]. Special header fields were introduced to request this action. If a draft contains the `Sign:` header field, `send` will initiate the signing. The signing key is either chosen automatically or it is specified by the `Pgpkey` profile entry. `send` always create signatures using the PGP/MIME standard [RFC 4880], but by invoking `mhsign` manually, old-style non-MIME signatures can be created as well. To encrypt an outgoing message, the draft needs to contain an `Enc:` header field. Public keys of all recipients are searched for in the gnupg keyring and in a file called `pgpkeys`, which contains exceptions and overrides. Unless public keys are found for all recipients, `mhsign` will refuse to encrypt it. Currently, messages with hidden (BCC) recipients can not be encrypted. This work is pending because it requires a structurally more complex approach.

`mhpgp` is the companion to `mhsign`. It verifies signatures and decrypts messages. Encrypted messages can be either temporarily decrypted and displayed or permanently decrypted and stored into the current folder. Currently, `mhpgp` needs to be invoked manually. The integration into `show` and `mhstore` to verify signatures and decrypt messages as needed is planned but not yet realized.

Both scripts were written for nmh. Hence they needed to be adjust according to the differences between nmh and mmh. For instance, they use the backup prefix no longer. Furthermore, compatibility support for old PGP features was dropped.

The integrated message signing and encrypting support is one of the most recent features in mmh. It has not had the time to mature. User feedback and personal experience need to be accumulated to direct the further development of the facility. Already it seems to be worthwhile to consider adding `-[no]sign` and `-[no]enc` switches to `send`, to be able to override the corresponding header fields. A profile entry:

```
send: -sign
```

would then activate signing for all outgoing messages. With the present approach, a `Send:` header component needs to be added to each draft template to achieve the same result. Adding the switches would ease the work greatly and keep the template files clean.

2.2.4 Draft and Trash Folder

Draft Folder

In the beginning, MH had the concept of a draft message. This was a file named `draft` in the MH directory, which was treated special. On composing a message, this draft file was used. When starting to compose another message before the former one was sent, the user had to decide among:

1. Using the old draft to finish and send it before starting with a new one.
2. Discarding the old draft and replacing it with a new one.
3. Preserving the old draft by refiling it to a folder.

Working on multiple drafts was only possible in alternation. For that, the current draft needed to be refiled to a folder and another one re-used for editing. Working on multiple drafts at the same time was impossible. The usual approach of switching to a different MH context did not help anything.

The draft folder facility exists to allow true parallel editing of drafts, in a straight forward way. It was introduced by Marshall T. Rose, already in 1984. Similar to other new features, the draft folder was inactive by default. Even in nmh, the highly useful draft folder was not available out-of-the-box. At least, Richard Coleman added the man page *mh-draft*(5) to better document the feature.

Not using the draft folder facility has the single advantage of having the draft file at a static location. This is simple in simple cases but the concept does not scale for more complex cases. The concept of the draft message is too limited for the problem it tries to solve. Therefore the draft folder was introduced. It is the more powerful and more natural concept. The draft folder is a folder like any other folder in MH. Its messages can be listed like any other messages. A draft message is no longer a special case. Tools do not need special switches to work on the draft message. Hence corner cases were removed.

The trivial part of the work was activating the draft folder with a default name. I chose the name `+drafts`, for obvious reasons. In consequence, the command line switches `-draftfolder` and `-draftmessage` could be removed. More difficult, but also more improving, was the updating of the tools to the new concept. By fully switching to the draft folder, the tools could be simplified by dropping the awkward draft message handling code. `-draft` switches were removed because operating on a draft message is no longer special. It became indistinguishable to operating on any other message. [☞ `337338b`]

There is no more need to query the user for draft handling [☞ `2d48b45`]. It is always possible to add another new draft. Refiling drafts is without difference to refiling other messages. All of these special cases are gone. Yet, one draft-related switch remained. `comp` still has `-[no]use` for switching between two modes:

1. Modifying an existing draft, with `-use`.
2. Composing a new draft, possibly taking some existing message as template, with `-nouse`, the default.

In either case, the behavior of `comp` is deterministic.

`send` now operates on the current message in the draft folder by default. As message and folder can both be overridden by specifying them on the command line, it is possible to send any message in the mail storage by simply specifying its number and folder. In contrast to the other tools, `send` takes the draft folder as its default folder.

Dropping the draft message concept in favor for the draft folder concept, replaced special cases with regular cases. This simplified the source code of the tools, as well as the concepts. In mmh, draft management does not break with the MH concepts but applies them. '`scan +drafts`', for instance, is a truly natural request.

Most of the work was already performed by Rose in the eighties. The original improvement of mmh is dropping the old draft message approach and thus simplifying the tools, the documentation, and the system as a whole. Although my part in the draft handling improvement was small, it was important.

Trash Folder

Similar to the situation for drafts is the situation for removed messages. Historically, a message was "deleted" by prepending a specific *backup prefix*, usually the comma character, to the file name. The specific file would then be ignored by MH because only files with names consisting of digits only are treated as messages. Although files remained in the file system, the messages were no longer visible in MH. To truly delete them, a maintenance job was needed. Usually a cron job was installed to delete them after a grace time. For instance:

```
find $HOME/Mail -type f -name ´,*´ -ctime +7 -delete
```

In such a setup, the original message could be restored within the grace time interval by stripping the backup prefix from the file name – usually but not always. If the last message of a folder with six messages (`1-6`) was removed, message `6`, became file `,6`. If then a new message entered the same folder, it would be named with the number one above the highest existing message number. In this case the message would be named `6`, reusing the number. If this new message would be removed as well, then the backup of the former message becomes overwritten. Hence, the ability to restore removed messages did not only depend on the sweeping cron job but also on the removing of further messages. It is undesirable to have such obscure and complex mechanisms. The user should be given a small set of clear assertions, such as "Removed files are restorable within a seven-day grace time." With the addition "... unless a message with the same name in the same folder is removed before." the statement becomes complex. A user will hardly be able to keep track of all removals to know if the assertion still holds true for a specific file. In practice, the real mechanism is unclear to the user.

Furthermore, the backup files were scattered within the whole mail storage. This complicated managing them. It was possible with the help of `find`, but everything is more convenient if the deleted messages are collected in one place.

The profile entry `rmmproc` (previously named `Delete-Prog`) was introduced very early to improve the situation. It could be set to any command, which would be executed to remove the specified messages. This had overridden the default action, described above. Refiling the to-be-removed files to a trash folder was the usual example. Nmh's man page *rmm*(1) proposes to set the `rmmproc` to '`refile +d`' to move messages to the trash folder `+d` instead of renaming them with the backup prefix. The man page additionally proposes the expunge command '`rm `mhpath +d all``' to empty the trash folder.

Removing messages in such a way has advantages:

1. The mail storage is prevented from being cluttered with removed messages because they are all collected in one place. Existing and removed messages are thus separated more strictly.
2. No backup files are silently overwritten.
3. Most important, however, removed messages are kept in the MH domain. Messages in the trash folder can be listed like those in any other folder. Deleted messages can be displayed like any other messages. `refile` can restore deleted messages. All operations on deleted files are still covered by the MH tools. The trash folder is just like any other folder in the mail storage.

Similar to the draft folder case, I dropped the old backup prefix approach in favor for replacing it by the better suiting trash folder system. Hence, `rmm` calls `refile` to move the to-be-removed message to the trash folder, `+trash` by default. To sweep it clean, the user can use '`rmm -unlink +trash a`', where the `-unlink` switch causes the files to be unlinked. [☞ `8edc5aa`] [☞ `ca0b3e8`]

Dropping the legacy approach and converting to the new approach completely, simplified the code base. The relationship between `rmm` and `refile` was inverted. In mmh, `rmm` invokes `refile`. That used to be the other way round. Yet, the relationship is simpler now. Loops, like described in nmh's man page for *refile*(1), can no longer occur:

> Since `refile` uses your `rmmproc` to delete the message, the `rmmproc` must NOT call `refile` without specifying `-normmproc` or you will create an infinite loop.

`rmm` either unlinks a message with `unlink()` or invokes `refile` to move it to the trash folder. `refile` does not invoke any tools.

By generalizing the message removal in the way that it became covered by the MH concepts made the whole system more powerful.

2.2.5 Modern Defaults

Nmh has a bunch of convenience-improving features inactive by default, although one can expect every new user to want them active. The reason they are inactive by default is the wish to stay compatible with old versions. But what are old versions? Still, the highly useful draft folder facility has not been activated by default although it was introduced over twenty-five years ago [Rose85]. The community seems not to care.

In nmh, new users are required to first build up a profile before they can access the modern features. Without an extensive profile, the setup is hardly usable for modern emailing. The point is not the customization of the setup, but the need to activate generally useful facilities. Yet, the real problem lies less in enabling the features, as this is straight forward as soon as one knows what he wants. The real problem is that new users need deep insight into the project to discover the available but inactive features. To give an example, I needed one year of using nmh before I became aware of the existence of the attachment system. One could argue that this fact disqualifies my reading of the documentation. If I would have installed nmh from source back then, I could agree. Yet, I had used a pre-packaged version and had expected that it would just work. Nevertheless, I had been convinced by the concepts of MH already and I am a software developer, still I required a lot of time to discover the cool features. How can we expect users to be even more advanced than me, just to enable them to use MH in a convenient and modern way? Unless they are strongly convinced of the concepts, they will fail. I have seen friends of me giving up disappointed before they truly used the system, although they had been motivated in the beginning. New users suffer hard enough to get used to the tool chest approach, we developers should spare them further inconveniences.

Maintaining compatibility for its own sake is bad, because the code base will collect more and more compatibility code. Sticking to the compatibility code means remaining limited; whereas adjusting to the changes renders the compatibility unnecessary. Keeping unused alternatives in the code for longer than a short grace time is a bad choice as they likely gather bugs by not being constantly tested. Also, the increased code size and the greater number of conditions increase the maintenance costs. If any MH implementation would be the back-end of widespread email clients with large user bases, compatibility would be more important. Yet, it appears as if this is not the case. Hence, compatibility is hardly important for technical reasons. Its importance originates from personal reasons rather. Nmh's user base is small and old. Changing the interfaces causes inconvenience to long-term users of MH. It forces them to change their many years old MH configurations. I do understand this aspect, but by sticking to the old users, new users are kept from entering the world of MH. But the future lies in new users. In consequence, mmh invites new users by providing a convenient and modern setup, readily usable out-of-the-box.

In mmh, all modern features are active by default and many previous approaches are removed or only accessible in a manual way. New default features include:

- The attachment system (`Attach:`) [☞ `8ff284f`].
- The draft folder facility (`+drafts`) [☞ `337338b`].
- The unseen sequence ('u') [☞ `c236056`] and the sequence negation prefix ('!') [☞ `db74c2b`].
- Quoting the original message in the reply [☞ `67411b1`].
- Forwarding messages using MIME [☞ `6e27160`].

An mmh setup with a profile that defines only the path to the mail storage, is already convenient to use. Again, Paul Vixie's supports the direction I took: ''the 'main branch' should just be modern'' [mail: edginess].

2.3 STYLING

Kernighan and Pike have emphasized the importance of style in the preface of *The Practice of Programming* [Kernighan99, p. x]:

> Chapter 1 discusses programming style. Good style is so important to good programming that we have chosen to cover it first.

This section covers changes in mmh that were guided by the desire to improve on style. Many of them follow the advice given in the quoted book.

2.3.1 Code Style

Indentation Style

Indentation styles are the holy cow of programming. Kernighan and Pike write [Kernighan99, p. 10]:

> Programmers have always argued about the layout of programs, but the specific style is much less important than its consistent application. Pick one style, preferably ours, use it consistently, and don't waste time arguing.

I agree that the constant application is most important, but I believe that some styles have advantages over others. For instance the indentation with tab characters only. The number of tabs corresponds to the nesting level – one tab, one level. Tab characters provide flexible visual appearance because developers can adjust their width as preferred. There is no more need to check for the correct mixture of tabs and spaces. Two simple rules ensure the integrity and flexibility of the visual appearance:

1. Leading whitespace must consist of tabs only.
2. All other whitespace should be spaces.

Although reformatting existing code should be avoided, I did it. I did not waste time arguing; I just reformatted the code. [☞ a485ed4]

Comments

Kernighan and Pike demand: ''Don't belabor the obvious'' [Kernighan99, p. 23]. Following the advice, I removed unnecessary comments. For instance, I removed all comments in the following code excerpt [☞ 4265436]:

```
context_replace(curfolder, folder);  /* update current folder  */
seq_setcur(mp, mp->lowsel);  /* update current message */
seq_save(mp);  /* synchronize message sequences */
folder_free(mp);  /* free folder/message structure */
context_save();  /* save the context file */

[...]

int c;  /* current character */
char *cp;  /* miscellaneous character pointer */

[...]

/* NUL-terminate the field */
*cp = '\0';
```

The information in each of the comments was present in the code statements already, except for the NUL-termination, which became obvious from the context.

Names

Regarding this topic, Kernighan and Pike suggest: “Use active names for functions” [Kernighan99, p. 4]. One application of this rule was the rename of `check_charset()` to `is_native_charset()` [☞ `8d77b48`]. The same change additionally fixed a violation of “Be accurate” [Kernighan99, p. 4], as the code did not match the expectation the function suggested. It did not compare charset names but prefixes of them only. In case the native charset was ‘ISO-8859-1’, then

```
check_charset("ISO-8859-11", strlen("ISO-8859-11"))
```

had returned true although the upper halves of the code pages are different.

More important than using active names is using descriptive names.

```
m_unknown(in);  /* the MAGIC invocation... */
```

Renaming the obscure `m_unknown()` function was a delightful event, although it made the code less funny [☞ `611d68d`].

Magic numbers are generally considered bad style. Obviously, Kernighan and Pike agree: “Give names to magic numbers” [Kernighan99, p. 19].

The argument `outnum` of the function `scan()` in `uip/scansbr.c` holds the number of the message to be created. As well it encodes program logic with negative numbers and zero. This led to obscure code. I clarified the code by introducing two variables that extracted the hidden information:

```
int incing = (outnum > 0);
int ismbox = (outnum != 0);
```

The readable names are thus used in conditions; the variable `outnum` is used only to extract ordinary message numbers [☞ `b8b075c`].

Through the clarity improvement of the change detours in the program logic of related code parts became apparent. The implementation was simplified. This possibility to improve had been invisible before [☞ `aa60b0a`].

The names just described were a first step, yet the situation was further improved by giving names to the magic values of `outnum`:

```
#define SCN_MBOX (-1)
#define SCN_FOLD 0
```

The two variables were updated thereafter as well:

```
int incing = (outnum != SCN_MBOX && outnum != SCN_FOLD);
int scanfolder = (outnum == SCN_FOLD);
```

Furthermore, `ismbox` was replaced by `scanfolder` because that matched better to the program logic. [☞ `7ffb36d`]

2.3.2 Structural Rework

Although the stylistic changes described already improve the readability of the source code, all of them were changes “in the small”. Structural changes, in contrast, affect much larger code areas. They are more difficult to accomplish but lead to larger improvements, especially as they often influence the outer shape of the tools as well.

At the end of their chapter on style, Kernighan and Pike ask: "But why worry about style?" [Kernighan99, p. 28]. Following are two examples of structural rework that demonstrate why style is important in the first place.

Rework of `anno`

Until 2002, `anno` had six functional command line switches: `-component` and `-text`, each with an argument, and the two pairs of flags, `-[no]date` and `-[no]inplace`. Then Jon Steinhart introduced his attachment system. In need for more advanced annotation handling, he extended `anno` and added five more switches: `-draft`, `-list`, `-delete`, `-append`, and `-number`, the last one taking an argument [☞ `7480dbc`]. Later, `-[no]preserve` was added as well [☞ `d9b1d57`]. Then, the Synopsis section of the man page *anno*(1) read:

```
anno [+folder] [msgs] [-component field] [-inplace | -noinplace]
     [-date | -nodate] [-draft] [-append] [-list] [-delete]
     [-number [num|all]] [-preserve | -nopreserve] [-version]
     [-help] [-text body]
```

The implementation followed the same structure. Problems became visible when '`anno -list -number 42`' worked on the current message instead of on message number 42, and '`anno -list -number l:5`' did not work on the last five messages but failed with the mysterious error message: "anno: missing argument to -list". Yet, the invocation matched the specification in the man page. There, the correct use of `-number` was defined as being '`[-number [num|all]]`' and the textual description for the combination with `-list` read:

> The `-list` option produces a listing of the field bodies for header fields with names matching the specified component, one per line. The listing is numbered, starting at 1, if the `-number` option is also used.

The problem was manifold. Semantically, the argument to the `-number` switch is only necessary in combination with `-delete`, but not with `-list`. The code, however, required a numeric argument in any case. If the argument was missing or non-numeric, `anno` aborted with an error message that additionally had an off-by-one error. It printed the name of the switch one before the concerned one.

Trying to fix these problems on the surface would not have solved them. They originate from a discrepance between the structure of the problem and the structure implemented in the program. Such structural differences can only be solved by adjusting the structure of the implementation to the structure of the problem.

Steinhart had added the `-list` and `-delete` switches in the same way as the other switches though they are of structural different type. Semantically, `-list` and `-delete` introduce operation modes. Historically, `anno` had only one operation mode: adding header fields. With the extension, two more modes were added: listing and deleting header fields. The structure of the code changes did not pay respect to this fundamental change. Neither the implementation nor the documentation did clearly declare the exclusive operation modes as such. Having identified the problem, I solved it by putting structure into `anno` and its documentation [☞ `d54c8db`].

The difference is visible in both the code and the documentation. For instance in the following code excerpt:

```
int delete = -2;  /* delete header element if set */
int list = 0;  /* list header elements if set */
[...]
     case DELETESW:  /* delete annotations */
          delete = 0;
          continue;
     case LISTSW:  /* produce a listing */
          list = 1;
          continue;
```

which was replaced by:

```
static enum { MODE_ADD, MODE_DEL, MODE_LIST } mode = MODE_ADD;
[...]
     case DELETESW:  /* delete annotations */
          mode = MODE_DEL;
          continue;
     case LISTSW:  /* produce a listing */
          mode = MODE_LIST;
          continue;
```

The replacement code does not only reflect the problem's structure better, it is easier to understand as well. The same applies to the documentation. The man page was completely reorganized to propagate the same structure. This is already visible in the Synopsis section:

```
anno [+folder] [msgs] [-component field] [-text body]
     [-append] [-date | -nodate] [-preserve | -nopreserve]
     [-Version] [-help]

anno -delete [+folder] [msgs] [-component field] [-text
     body] [-number num | all ] [-preserve | -nopreserve]
     [-Version] [-help]

anno -list [+folder] [msgs] [-component field] [-number]
     [-Version] [-help]
```

Path Conversion

Four kinds of path names can appear in MH:

1. Absolute Unix directory paths, like `/etc/passwd`.
2. Relative Unix directory paths, like `./foo/bar`.
3. Absolute MH folder paths, like `+projects/mmh`.
4. Relative MH folder paths, like `@subfolder`.

Relative MH folder paths, are hardly documented although they are useful for large mail storages. The current mail folder is specified as '`@`', just like the current directory is specified as '`.`'.

To allow MH tools to understand all four notations, they need to be able to convert between them. In nmh, these path name conversion functions were located in the files `sbr/path.c` ("return a pathname") and `sbr/m_maildir.c` ("get the path for the mail directory"). The seven functions in the two files were documented with no more than two comments, which described obvious information. The signatures of the four exported functions did not explain their semantics:

1. `char *path(char *, int);`
2. `char *pluspath(char *);`
3. `char *m_mailpath(char *);`
4. `char *m_maildir(char *);`

My investigations provided the following descriptions:

1. The second parameter of `path()` defines the type as which the path given in the first parameter should be treated. Directory paths are converted to absolute directory paths. Folder paths are converted to absolute folder paths. Folder paths must not include a leading '`@`' character. Leading plus characters are preserved. The result is a pointer to newly allocated memory.
2. `pluspath()` is a convenience-wrapper to `path()`, to convert folder paths only. This function can not be used for directory paths. An empty string parameter causes a buffer overflow.
3. `m_mailpath()` converts directory paths to absolute directory paths. The characters '`+`' or '`@`' at the beginning of the path name are treated literal, i.e. as the first character of a relative directory path. Hence, this function can not be used for folder paths. In any case, the result is an absolute directory path, returned as a pointer to newly allocated memory.
4. `m_maildir()` returns the parameter unchanged if it is an absolute directory path or begins with the entry '`.`' or '`..`'. All other strings are prepended with the current working directory. Hence, this function can not be used for folder paths. The result is either an absolute directory path or a relative directory path, starting with dot or dot-dot. In contrast to the other functions, the result is a pointer to static memory.

The situation was obscure, irritating, error-prone, and non-orthogonal. Additionally, no clear terminology was used to name the different kinds of path names. Sometimes, the names were even misleading, much as the first argument of `m_mailpath()`, which was named `folder`, although `m_mailpath()` could not be used with MH folder arguments.

I clarified the path name conversion by complete rework. First of all, the terminology needed to be defined. A path name is either in the Unix domain, then it is called *directory path* or it is in the MH domain, then it is called *folder path*. The two terms need to be used with strict distinction. Second, I exploited the concept of path type indicators. By requiring every path name to start with a distinct type identifier, the conversion between the types could be fully automated. This allows the tools to accept path names of any type from the user. Therefore, it was necessary to require relative directory paths to be prefixed with a dot character. In consequence, the dot character could no longer be an alias for the current message [☞ cff0e16]. Third, I created three new functions to replace the previous mess:

1. `expandfol()` converts folder paths to absolute folder paths. Directory paths are simply passed through. This function is to be used for folder paths only, thus the name. The result is a pointer to static memory.
2. `expanddir()` converts directory paths to absolute directory paths. Folder paths are treated as relative directory paths. This function is to be used for directory paths only, thus the name. The result is a pointer to static memory.
3. `toabsdir()` converts any type of path to an absolute directory path. This is the function of choice for path conversion. Absolute directory paths are the most general representation of a path name. The result is a pointer to static memory.

The new functions have names that indicate their use. Two of the functions convert relative to absolute path names of the same type. The third function converts any path name type to the most general one, the absolute directory path. All of the functions return pointers to static memory. The file `sbr/path.c` contains the implementation of the functions; `sbr/m_maildir.c` was removed. [☞ d39e2c4]

Along with the path conversion rework, I also replaced `getfolder(FDEF)` with `getdeffol()` and `getfolder(FCUR)` with `getcurfol()`, which only wraps `expandfol("@")` for convenience. This code was moved from `sbr/getfolder.c` into `sbr/path.c` as well. [☞ d39e2c4]

The related function `etcpath()` is now included in `sbr/path.c`, too [☞ b4c2979]. Previously, it had been located in `config/config.c`.

Now, `sbr/path.c` contains all path handling code. Besides being less code, its readability is highly improved. The functions follow a common style and are well documented.

2.3.3 Profile Reading

The MH profile contains the configuration of a user-specific MH setup. MH tools read the profile right after starting up because it contains the location of the user's mail storage and similar settings that influence the whole setup. Furthermore, the profile contains the default switches for the tools as well. The context file is read along with the profile.

For historic reasons, some MH tools did not read the profile and context. Among them were `post`/`spost`, `mhmail`, and `slocal`. The reason why these tools ignored the profile were not clearly stated. During a discussion on the nmh-workers mailing list, David Levine posted an explanation, quoting John Romine [mail: levine]:

> I asked John Romine and here's what he had to say, which agrees and provides an example that convinces me:
>
> > My take on this is that `post` should not be called by users directly, and it doesn't read the `.mh_profile` (only front-end UI programs read the profile).

> > For example, there can be contexts where post is called by a helper program (like 'mhmail') which may be run by a non-MH user. We don't want this to prompt the user to create an MH profile, etc.
> >
> > My suggestion would be to have send pass a (hidden) '-fileproc *proc*' option to post if needed. You could also use an environment variable (I think send/whatnow do this).
>
> I think that's the way to go. My personal preference is to use a command line option, not an environment variable.

To solve the problem that post does not honor the fileproc profile entry, the community roughly agreed that a switch -fileproc should be added to post to be able to pass a different fileproc. I strongly disagree with this approach because it does not solve the problem; it only removes a single symptom. The actual problem is that post does not behave as expected, though all programs should behave as expected. Clear and general concepts are a precondition for this. Thus, there should be no separation into "front-end UI programs" and ones that "should not be called by users directly". The real solution is having all MH tools read the profile.

But the problem has a further aspect, which originates from mhmail mainly. mhmail was intended to be a replacement for mailx on systems with MH installations. In difference to mailx, mhmail used MH's post to send the message. The idea was that using mhmail should not be influenced whether the user had MH set up for himself or not. Therefore mhmail had not read the profile. As mhmail used post, post was not allowed to read the profile neither. This is the reason for the actual problem. Yet, this was not considered much of a problem because post was not intended to be used by users directly. To invoke post, send was used an a front-end. send read the profile and passed all relevant values on the command line to post – an awkward solution.

The important insight is that mhmail is a wolf in sheep's clothing. This alien tool broke the concepts because it was treated like a normal MH tool. Instead it should have been treated accordingly to its foreign style.

The solution is not to prevent the tools from reading the profile but to instruct them to read a different profile. mhmail could have set up a well-defined profile and caused the following post to use this profile by exporting an environment variable. With this approach, no special cases would have been introduced and no surprises would have been caused. By writing a wrapper program to provide a clean temporary profile, the concept could have been generalized orthogonally to the whole MH tool chest.

In mmh, the wish to have mhmail as a replacement for mailx is considered obsolete. Mmh's mhmail does no longer cover this use-case [☞ d36e56e]. Currently, mhmail is in a transition state [☞ 32d4f9d]. It may become a front-end to comp, which provides an alternative interface which can be more convenient in some cases. This would convert mhmail into an ordinary MH tool. If, however, this idea does not convince, then mhmail will be removed.

In the mmh tool chest, every program reads the profile. (slocal is not considered part of the mmh tool chest (cf. Sec. 2.1.2).) Mmh has no post program, but it has spost, which now does read the profile [☞ 3e017a7]. Following this change, send and spost can be considered for merging. Besides send, spost is only invoked directly by the to-be-changed mhmail implementation and by rcvdist, which requires rework anyway.

Jeffrey Honig quoted Marshall T. Rose explaining the decision that post ignores the profile [mail: honig]:

> when you run mh commands in a script, you want all the defaults to be what the man page says. when you run a command by hand, then you want your own defaults...

The explanation neither matches the problem concerned exactly nor is the interpretation clear. If the described desire addresses the technical level, then it conflicts with the Unix philosophy, precisely because the indistinquishability of human and script input is the main reason for the huge software leverage in Unix. If, however, the described desire addresses the user's view, then different technical solutions are more appropriate. The two cases can be regarded simply as two different MH setups. Hence, mapping the problem of different behavior between interactive and automated use on the concept of switching between different profiles, marks it already solved.

2.3.4 Standard Libraries

MH is one decade older than the POSIX and ANSI C standards. Hence, MH included own implementations of functions that were neither standardized nor widely available, back then. Today, twenty years after POSIX and ANSI C were published, developers can expect that systems comply with these standards. In consequence, MH-specific replacements for standard functions can and should be dropped. Kernighan and Pike advise: "Use standard libraries" [Kernighan99, p. 196]. Actually, MH had followed this advice in history, but it had not adjusted to more recent changes in this field. The snprintf() function, for instance, was standardized with C99 and is available almost everywhere because of its high usefulness. Thus, the project's own implementation of snprintf() was dropped in March 2012 in favor for using the one of the standard library [☞ 0052f10]. Such decisions limit the portability of mmh if systems do not support these standardized and widespread functions. This compromise is made because mmh focuses on the future.

As I am still in my twenties, have no programming experience from past decades. I have not followed the evolution of C through time. I have not suffered from the the Unix wars. I have not longed for standardization. All my programming experience is from a time when ANSI C and POSIX were well established already. Thus, I needed to learn about the history in retrospective. I have only read a lot of books about the (good) old times. This put me in a difficult position when working with old code. I need to freshly acquire knowledge about old code constructs and ancient programming styles, whereas older programmers know these things by heart from their own experience. Being aware of the situation, I rather let people with more historic experience do the transition from ancient code constructs to standardized ones. Lyndon Nerenberg covered large parts of this task for the nmh project. He converted project-specific functions to POSIX replacements, also removing the conditionals compilation of now standardized features. Ken Hornstein and David Levine had their part in this work, as well. Often, I only pulled the changes over from nmh into mmh. These changes include many commits, among them: [☞ 768b5ed] [☞ 0052f10].

Nevertheless, I worked on the task as well, tidying up the *MH standard library*, `libmh.a`. It is located in the `sbr` ("subroutines") directory in the source tree and includes functions that mmh tools usually need. Among them are MH-specific functions for profile, context, sequence, and folder handling, but as well MH-independent functions, such as auxiliary string functions, portability interfaces and error-checking wrappers for critical functions of the standard library.

- I have replaced the `atooi()` function with calls to `strtoul()`, setting the third parameter, the base, to eight. `strtoul()` is part of C89 and thus considered safe to use [☞ `c490c51`].
- I did remove project-included fallback implementations of `memmove()` and `strerror()` [☞ `b067ff5`], although Peter Maydell had re-included them into nmh in 2008 to support SunOS 4. Nevertheless, these functions are part of ANSI C. Systems that do not even provide full ANSI C support should not put a load on mmh.
- The `copy()` function copies the string in parameter one to the location in parameter two. In contrast to `strcpy()`, it returns a pointer to the terminating null-byte in the destination area. The code was adjusted to replace `copy()` with `strcpy()`, except within `concat()`, where `copy()` was more convenient. Therefore, the definition of `copy()` was moved into the source file of `concat()` and its visibility it limited to that [☞ `552fd72`].
- The function `r1bindex()` had been a generalized version of `basename()` with minor differences. As all calls to `r1bindex()` had the slash ('`/`') as delimiter anyway, replacing `r1bindex()` with the more specific and better-named function `basename()` became desirable. Unfortunately, many of the 54 calls to `r1bindex()` depended on a special behavior, which differed from the POSIX specification for `basename()`. Hence, `r1bindex()` was kept but renamed to `mhbasename()`, setting the delimiter to the slash [☞ `2400138`]. For possible uses of `r1bindex()` with a different delimiter, the ANSI C function `strrchr()` provides the core functionality.
- The `ssequal()` function – apparently for "substring equal" – was renamed to `isprefix()`, because this is what it actually checked [☞ `c20b4fa`] Its source file had included both of the following comments, no joke.

  ```
  /*
   * THIS CODE DOES NOT WORK AS ADVERTISED.
   * It is actually checking if s1 is a PREFIX of s2.
   * All calls to this function need to be checked to see
   * if that needs to be changed. Prefix checking is cheaper, so
   * should be kept if it´s sufficient.
   */

  /*
   * Check if s1 is a substring of s2.
   * If yes, then return 1, else return 0.
   */
  ```

  Eventually, the function was completely replaced with calls to `strncmp()` [☞ `b0b1dd3`].

2.3.5 User Data Locations

In nmh, a personal setup consists of the MH profile and the MH directory. The profile is a file named `.mh_profile` in the user's home directory. It contains the static configuration. It also contains the location of the MH directory in the profile entry `Path`. The MH directory contains the mail storage and is the first place to search for form files, scan formats, and similar configuration files. The location of the MH directory can be chosen freely by the user. The usual name is a directory named `Mail` in the user's home directory.

The way MH data is split between profile and MH directory is a legacy. It is only sensible in a situation where the profile is the only configuration file. Why else should the mail storage and the configuration files be intermixed? They are of different kind: One kind is the data to be operated on and the other kind is the configuration to change how tools operate. Splitting the configuration between the profile and the MH directory is inappropriate, as well. I improved the situation by breaking compatibility.

In mmh, personal data is grouped by type. This results in two distinct parts: the mail storage and the configuration. The mail storage directory still contains all the messages, but, in exception of public sequences files, nothing else. In difference to nmh, the auxiliary configuration files are no longer located there. Therefore, the directory is no longer called the user's *MH directory* but the user's *mail storage*. Its location is still user-chosen, with the default name `Mail` in the user's home directory. The configuration is grouped together in the hidden directory `.mmh` in the user's home directory. This *mmh directory* contains the context file, personal forms, scan formats, and the like, but also the user's profile, now named `profile`. The path to the profile is no longer `$HOME/.mh_profile` but `$HOME/.mmh/profile`. (The alternative of having file `$HOME/.mh_profile` and a configuration directory `$HOME/.mmh` appeared to be inconsistent.)

The approach chosen for mmh is consistent, simple, and familiar to Unix users. The main achievement of the change is the clear and sensible separation of the mail storage and the configuration. [☞ `7030d7e`]

As MH allows users to have multiple MH setups, it is necessary to switch the profile. The profile is the single entry point to access the rest of a personal MH setup. In nmh, the environment variable `MH` is used to specify a different profile. To operate in the same MH setup with a separate context, the `MHCONTEXT` environment variable is used. This allows having a separate current folder in each terminal at the same time, for instance. In mmh, three environment variables replace the two of nmh. `MMH` overrides the default location of the mmh directory (`.mmh`). `MMHP` and `MMHC` override the paths to the profile and context file, respectively. This approach allows the set of personal configuration files to be chosen independently of the profile, context, and mail storage. The new approach has no functional disadvantages, as every setup I can imagine can be implemented with both approaches, possibly even easier with the new one. [☞ `7030d7e`]

2.3.6 Modularization

The source code of the mmh tools is located in the `uip` ("user interface programs") directory. Each tool has a source file with the name of the command. For example, `rmm` is built from uip/rmm.c. Some source files are used for multiple programs. For example uip/scansbr.c is used for both scan and inc. In nmh, 49 tools were built from 76 source files. This is a ratio of 1.6 source files per program. 32 programs depended on multiple source files; 17 programs depended on one source file only. In mmh, 39 tools are built from 51 source files. This is a ratio of 1.3 source files per program. 18 programs depend on multiple source files; 21 programs depend on one source file only. (These numbers and the ones in the following text ignore the MH library as well as shell scripts and multiple names for the same program.)

Splitting the source code of a large program into multiple files can increase the readability of its source code, but most of the mmh tools are small and straightforward programs. In exception of the MIME handling tools (i.e. mhbuild, mhstore, show, etc.), pick is the only tool with more than one thousand lines of source code. Splitting programs with less than one thousand lines of code into multiple source files leads seldom to better readability. For such tools, splitting still makes sense when parts of the code are reused in other programs and the reused code fragment is (1) not general enough for including it in the MH library or (2) has dependencies on a library that only few programs need. uip/packsbr.c, for instance, provides the core program logic for the packf and rcvpack programs. uip/packf.c and uip/rcvpack.c mainly wrap the core function appropriately. No other tools use the folder packing functions. As another example, uip/termsbr.c accesses terminal properties, which requires linking with the *termcap* or a *curses* library. If uip/termsbr.c is included in the MH library, then every program needs to be linked with termcap or curses, although only few of the programs use the library.

The task of MIME handling is complex enough that splitting its code into multiple source files improves the readability. The program mhstore, for instance, is compiled out of seven source files with 2 500 lines of code in summary. The main code file uip/mhstore.c consists of 800 lines; the other 1 700 lines are code reused in other MIME handling tools. It seems to be worthwhile to bundle the generic MIME handling code into a MH-MIME library, as a companion to the MH standard library. This is left to be done.

The work already accomplished focussed on the non-MIME tools. The amount of code compiled into each program was reduced. This eases the understanding of the code base. In nmh, comp was built from six source files: comp.c, whatnowproc.c, whatnowsbr.c, sendsbr.c, annosbr.c, and distsbr.c. In mmh, it builds from only two: comp.c and whatnowproc.c. In nmh's comp, the core function of whatnow, send, and anno were all compiled into comp. This saved the need to execute these programs with the expensive system calls fork() and exec(). Whereas this approach improved the time performance, it interwove the source code. Core functionalities were not encapsulated into programs but into function, which were then wrapped by programs. For example, uip/annosbr.c included the function annotate(). Each program that wanted to annotate messages, included the source file uip/annosbr.c and called annotate(). Because the function annotate() was used like the tool anno, it had seven parameters, reflecting the command line switches of the tool. When another pair of command line switches was added to anno, a rather ugly hack was implemented to avoid adding another parameter to the function [☞ d9b1d57].

In mmh, the relevant code of comp comprises the two files uip/comp.c and uip/whatnowproc.c, together 210 lines of code, whereas in nmh, comp comprises six files with 2 450 lines. Not all of the code in these six files is actually used by comp, but the reader needed to read it all to know which parts are relevant. Understanding nmh's comp, required understanding the inner workings of uip/annosbr.c first. To be sure to fully understand a program, its whole source code needs to be examined. Not doing so is a leap of faith, assuming that the developers have avoided obscure programming techniques. Here, it should be recalled that information passed in obscure ways through the program's source base, due to the aforementioned hack to save an additional parameter in nmh's anno.

In mmh, understanding comp requires to read only 210 lines of code to read, whereas the amount is ten times more for nmh's comp.

By separating the tools on the program-level, the boundaries are clearly visible, as the interfaces are calls to exec() rather than arbitrary function calls. Additionally, this kind of separation is more strict because it is technically enforced by the operating system; it can not be simply bypassed with global variables. Good separation simplifies the understanding of program code because the area influenced by any particular statement is small. As I have read a lot in nmh's code base during the last two years, I have learned about the easy and the difficult parts. In my observation, the understanding of code is enormously eased if the influenced area is small and clearly bounded.

Yet, the real problem is another: Nmh violates the golden "one tool, one job" rule of the Unix philosophy. Understanding comp requires understanding uip/annosbr.c and uip/sendsbr.c because comp annotates and sends messages. In nmh, there surely exist the tools anno and send, which cover these jobs, but comp and repl and forw and dist and whatnow and viamail – they all (!) – have the same annotating and sending functions included, once more. As a result, comp sends messages without using send. The situation is the same as if grep would page its output without using more just because both programs are part of the same code base.

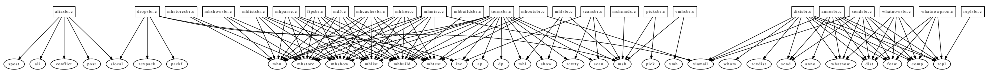
The clear separation on the surface of nmh – the tool chest approach – is violated on the level below. This violation is for the sake of time performance. Decades ago, sacrificing readability and conceptional beauty for speed might have been necessary to prevent MH from being unusably slow, but today this is not the case anymore. No longer should speed improvements that became unnecessary be kept. No longer should readability or conceptional beauty be sacrificed. No longer should the Unix philosophy's "one tool, one job" guideline be violated. Therefore, mmh's comp no longer sends messages.

In mmh, different jobs are divided among separate programs that invoke each other as needed. In consequence, comp invokes whatnow which thereafter invokes send [☞ 3df5ab3] [☞ c73c00b]. The clear separation on the surface is maintained on the level below. Human users and other tools use the same interface – annotations, for example, are made by invoking anno, no matter if requested by programs or by human beings [☞ 469a416] [☞ aed3841] [☞ 3caf9e2].

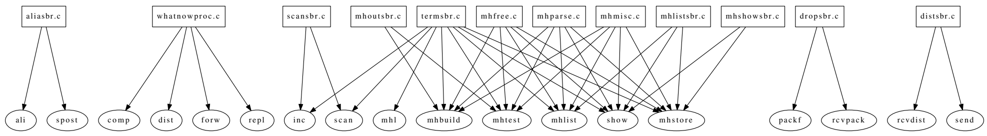
The decrease of tools built from multiple source files and thus the decrease of uip/*sbr.c files confirm the improvement [☞ 9e6d913] [☞ f0f8580] [☞ 0503a6e]

[☞ 27826f9] [☞ d1da1f9] [☞ c422228]. This is also visible in the complexity of the build dependency graphs:

Nmh:



Mmh:



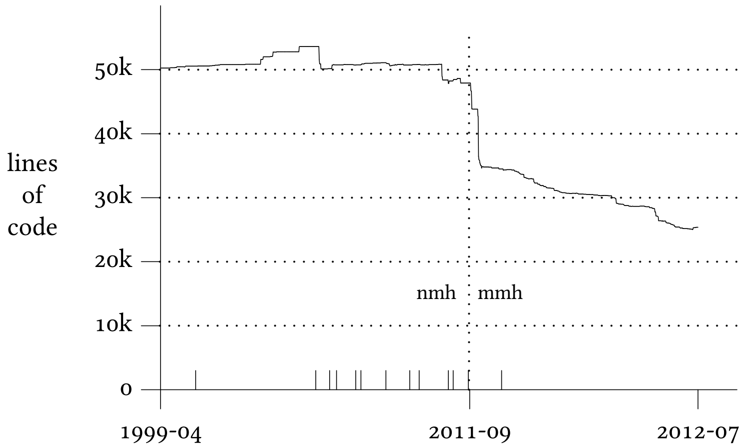
The figures display all program to source file relationships where programs (ellipses) are built from multiple source files (rectangles). The primary source file of each program is omited from the graph.

Chapter 3
SUMMARY

This document describes and explains my work on mmh. I have streamlined the project by removing programs, facilities, and options that diverted from the main task of mmh, being an MUA. I have modernized the code base removing obsolete functions and activating modern features per default. Furthermore, I have improved the style by refactoring clumpsy code and by identifying and forcing clear concepts. All my work was motivated by Antoine de Saint Exupéry's well-known statement [Saint-Exupéry39]:

> It seems that perfection is attained not when there is nothing more to add,
> but when there is nothing more to remove.

Against the common expectations, I hardly added new features. I regard my achievement in the selection of the relevant set of existing features, the choice of sensible defaults, and the extensive focus on structure and concepts. I believe, the result is a system simpler and clearer for both developing and using, without lacking important functionality.



Changesets in the version control system, in linear scale.
(The dates and the inner ticks, which mark the
turns of the years, are for convenience only.)

Outlook

MIME handling is the most complex part of mmh and the one with the highest potential for improvements. The changes already accomplished so far build upon the existing structure, but deeper rework is necessary to integrate MIME handling consistently. For instance, accessing messages and accessing their MIME parts should both covered by a single approach. This requires the sequence notation to provide a way to address MIME parts directly. In general, the sequence notation should become more powerful. For instance, it is currently not possible to access the second last message in a given sequence. Furthermore, displaying messages can be improved. Encrypted messages should be decoded automatically and digital signatures verified on-the-fly. In this rework, MH's unique features need to be preserved, but as well the default behavior should become less surprising. Still, encoding and decoding is not done everywhere it is necessary. The problems of not decoded quotations of the original message in replies and not encoded non-ASCII characters in the message header remain.

Some of mmh's tools were hardly touched, yet. Among them are `dist`, `rcvdist`, `mark`, `pick`, and `sortm`. They should be refactored as well. Related to `sortm` is the threaded message view, which is completely missing, so far. `pick` could profit from message indexing. These fields deserve further research.

Nmh's testing framework has not been updated for mmh, yet. All refactoring had been done without the safety net of a test framework. Hence, experience warns that there may be subtle bugs in the code base. As a consequence of the modularization rework (cf. Sec. 2.3.6), the compiler can no longer check the integrity of the interfaces when tools invoke each other. Automated testing should detect errors there.

The features most often asked for are IMAP and Maildir support. But, both of them collide with MH in the same fundamental way as different file system approaches collide with Unix. Nevertheless, an abstraction layer could provide a mapping between such storage back-ends and the MH storage format. Or, the mmh tool chest could be reworked to operate on a generic back-end, making the MH storage format only one of many possible back-ends. Research in this area is highly appreciated.

Relationship to nmh

The mmh project started as an experimental version of nmh because the nmh community did not welcome my plans and visions. The need to convincing the community of every change I liked to undertake would have slowed down my work too much. Hence, I created this experimental version to convince by demonstration.

While I worked on mmh, nmh's community became very active as well. Although we both worked on the same code base, there was no collaboration. This, I must admit, was my failure because I kept my work hidden from the nmh community. The reasons are personal and community-related. I am sorry for that and I like to improve in the future. Nonetheless, I did not work behind completely closed doors. I discussed within the regional computer community and presented the project in two video-recorded lectures [lecture: cs, lecture: gpn]. First users appeared and provided feedback.

Over time, I had to realize that, although nmh and mmh have much in common, the projects target different goals. I am still undecided how to handle it, but my experimental version more and more feels like being a fork. As I am strongly convinced that the path taken for the development of mmh is a good one, I like to push the project farther in this direction.

Appendix A
TOOLS OF MMH

Tool	Description	Part of nmh	Part of mmh	Amount of change
`ali`	list mail aliases	√	√	**
`anno`	annotate messages	√	√	***
`ap`	parse addresses 822-style	√	√	*
`burst`	explode digests into messages	√	√	**
`comp`	compose a message	√	√	**
`dist`	redistribute a message to additional addresses	√	√	—
`dp`	parse dates 822-style	√	√	*
`flist`	list folders with messages in given sequence	√	√	—
`flists`	list all folders with messages in given sequence	√	√	—
`fmtdump`	decode mmh format files	√	√	—
`fnext`	change to next folder with new messages	√	√	—
`folder`	set/list current folder/message	√	√	*
`folders`	list all folders	√	√	*
`forw`	forward messages	√	√	***
`fprev`	change to previous folder with new messages	√	√	—
`inc`	incorporate new mail	√	√	*
`mark`	mark messages	√	√	—
`mhbuild`	translate MIME composition draft	√	√	**
`mhlist`	list information about content of MIME messages	√	√	**
`mhl`	produce formatted listings of mmh messages	√	√	***
`mhmail`	send mail (mailx replacement)	√	√	***
`mhparam`	print mmh profile components	√	√	*
`mhpath`	print full pathnames of mmh messages and folders	√	√	—
`mhpgp`	verify and decrypt a message with gnupg	—	√	
`mhsign`	sign or encrypt a message with gnupg	—	√	
`mhstore`	store contents of MIME messages into files	√	√	**
`mmh`	initialize the mmh environment	—	√	
`mmhwrap`	adjust the search path	—	√	
`new`	report on folders with new messages	√	√	—
`next`	show the next message	√	√	*
`packf`	pack a folder into mbox format	√	√	***
`pick`	select messages by content	√	√	—
`prev`	show the previous message	√	√	*
`prompter`	prompting editor front end	√	√	**
`rcvdist`	asynchronously redistribute new mail	√	√	—
`rcvpack`	asynchronously append a message to an mbox file	√	√	**

Tool	Description	Part of nmh	Part of mmh	Amount of change
`rcvstore`	asynchronously incorporate new mail	√	√	*
`refile`	file messages in other folders	√	√	***
`repl`	reply to a message	√	√	**
`rmf`	remove folder	√	√	*
`rmm`	remove messages	√	√	***
`scan`	produce a one line per message scan listing	√	√	**
`sendfiles`	send multiple files in a MIME message	√	√	***
`send`	send a message	√	√	***
`show`	display MIME messages (renamed from `mhshow`)	√	√	***
`slocal`	asynchronously filter and deliver new mail	√	√	**
`sortm`	sort messages	√	√	–
`spost`	deliver a message	√	√	***
`unseen`	scan new messages in all folders	√	√	–
`whatnow`	prompting front-end for send	√	√	**
`whom`	list recipients of a message	√	√	***

Nmh tools removed from mmh:

`conflict`	search for alias/password conflicts	√	–	
`install-mh`	initialize the nmh environment	√	–	
`mhn`	display/list/store/cache MIME messages	√	–	
`msgchk`	check for messages	√	–	
`msh`	nmh shell (and BBoard reader)	√	–	
`post`	deliver a message	√	–	
`rcvtty`	report new mail	√	–	
`show`	display messages (`mhshow` in now known as `show`)	√	–	

REFERENCES

Anderson89

Robert H. Anderson, Norman Z. Shapiro, Tora K. Bikson, and Phyllis H. Kantar, ''The Design of the MH Mail System,'' (N-3017-IRIS), The RAND Corporation, December 1989.

Bourne83

Stephen R. Bourne, *The UNIX System,* International Computer Science Series, Addison-Wesley, 1983. ISBN: 0-201-13791-7.

Brooks86

Frederick P. Brooks, Jr., ''No Silver Bullet: Essence and Accidents of Software Engineering,'' in *Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference*, p. 1069–1076, Elsevier Science B.V., Amsterdam, The Netherlands, 1986.

Costales08

Bryan Costales, Claus Aßmann, George Jansen, and Gregory N. Shapiro, *sendmail,* Fourth Edition, O'Reilly, 2008. ISBN: 0-596-51029-2.

Curry96

David A. Curry, *UNIX Systems Programming for SVR4,* Nutshell Series, O'Reilly, 1996. ISBN: 1-56592-163-1.

Gancarz95

Mike Gancarz, *The UNIX Philosophy,* Digital Press, 1995. ISBN: 1-55558-123-4.

Hegardt90

Mary Hegardt and Tim Morgan, *MH for Beginners*, April 12 1990. http://git.savannah.gnu.org/cgit/nmh.git/plain/docs/historical/beginners.pdf

Kernighan84

Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment,* Prentice Hall, 1984. ISBN: 0-13-937681-X.

Kernighan88

Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language,* Second Edition, Prentice Hall, 1988. ISBN: 0-13-110362-8.

Kernighan99

Brian W. Kernighan and Rob Pike, *The Practice of Programming,* Addison-Wesley, 1999. ISBN: 0-201-61586-X.

lecture: cs

Lecture: Markus Schnalke, *mmh,* ChaosSeminar, CCC Ulm, April 2012. http://ulm.ccc.de/ChaosSeminar/2012/04_mmh

lecture: gpn

Lecture: Markus Schnalke, *meillo's mail handler,* Gulaschprogrammiernacht, Entropia Karlsruhe, June 2012. https://entropia.de/GPN12:meillo%27s_mail_handler

mail: attach
Markus Schnalke, “[patch] snapshot of my MIME handling improvments,” *nmh-workers mailing list*, Message-ID: <1PH2kD-0qV-00@serveme.schnalke.local>, November 13 2010. http://lists.nongnu.org/archive/html/nmh-workers/2010-11/msg00111.html

mail: edginess
Paul Vixie, “edginess,” *nmh-workers mailing list*, Message-ID: <4EF8B937.6060107@isc.org>, December 26 2011. http://lists.nongnu.org/archive/html/nmh-workers/2011-12/msg00117.html

mail: honig
Jeffrey Honig, “nmh ‘post’ doesn’t read profile, fileproc ignored,” *nmh-workers mailing list*, Message-ID: <CAC5WE8o+iMr3WXyUAVtp-b611+3w6sJS+-dc_qtQSXywNuzp9g@mail.gmail.com>, January 2 2012. http://lists.nongnu.org/archive/html/nmh-workers/2012-01/msg00005.html

mail: levine
David Levine, “nmh ‘post’ doesn’t read profile, fileproc ignored,” *nmh-workers mailing list*, Message-ID: <24209.1325713661@nist-cbox32.combinenet.com>, January 4 2012. http://lists.nongnu.org/archive/html/nmh-workers/2012-01/msg00016.html

mail: mmh-ann
Markus Schnalke, “Experimental version: mmh,” *nmh-workers mailing list*, Message-ID: <1RYetl-4Ok-00@serveme.home.schnalke.org>, December 8 2011. http://lists.nongnu.org/archive/html/nmh-workers/2011-12/msg00038.html

mail: mta-mua
Thread with the subjects: “nmh @ gsoc”, “external MTA” and “should nmh be an MTA or an MUA?,” *nmh-workers mailing list*, January 2010. http://lists.nongnu.org/archive/html/nmh-workers/2010-01/msg00026.html

mail: nmh-goal
Markus Schnalke, “Understanding nmh (aka. What’s the goal),” *nmh-workers mailing list*, Message-ID: <1PNSw4-12C-00@serveme.schnalke.local>, November 30 2010. http://lists.nongnu.org/archive/html/nmh-workers/2010-11/msg00193.html

McIlroy78
M. D. McIlroy, E. N. Pinson, and B. A. Tague, “UNIX Time-Sharing System: Foreword,” *The Bell System Technical Journal*, vol. 57, no. 6, p. 1902, Bell Laboratories, 1978.

MH-Memo
The Original MH-Proposal Memorandum: Stock Gaines and Norm Shapiro, *The Next Message System*, RAND Corporation, Undated. Unpublished. Quoted in *RAND and the Information Evolution* by Willis H. Ware, 2008, p. 129 ff. Also available online at: http://rand-mh.sourceforge.net/book/overall/hiofmh.html#TOMHP

Moss88
Sara E. Moss and Purvis M. Jackson, “An AJPO User’s Guide for MH, the Rand Message Handling System,” (CMU/SEI-88-UG-1, ESD-TR-88-030), Carnegie Mellon University, September 1988. http://www.dtic.mil/dtic/tr/fulltext/u2/a204635.pdf

Peek95
: Jerry Peek, *MH & xmh: Email for Users & Programmers,* O'Reilly, 1995. An updated version of the book (named *MH & nmh*) is freely available on the Internet: `http://rand-mh.sourceforge.net/book/`. The latest update happened in May 2006.

Raymond04
: Eric S. Raymond, *The Art of UNIX Programming,* Addison-Wesley, 2004. ISBN: 0-13-142901-9. Also available online at: `http://www.faqs.org/docs/artu/`

Rochkind85
: Marc J. Rochkind, *Advanced UNIX Programming,* Software Series, Prentice-Hall, 1985. ISBN: 0-13-011800-1.

Rose85
: Marshall T. Rose and John L. Romine, "MH.5: How to process 200 messages a day and still get some real work done," in *Proceedings, Summer Usenix Conference and Exhibition*, p. 455–487, Portland, Oregon, June 1985.

Rose86
: Marshall T. Rose and Jerry N. Sweet, *The Rand MH Message Handling System: Tutorial*, May 21 1986. `http://git.savannah.gnu.org/cgit/nmh.git/plain/docs/historical/tutorial.pdf`

Saint-Exupéry39
: Antoine de Saint-Exupéry, *Wind, Sand and Stars,* Reynal & Hitchcock, New York, 1939.

Salus94
: Peter H. Salus, *A Quarter Century of UNIX,* Addison-Wesley, 1994. ISBN: 0-201-54777-5.

Schnalke10
: Markus Schnalke, "Why the Unix Philosophy still matters," Term paper, Ulm University, 2010. `http://marmaro.de/docs/studium/unix-phil/`

Sill02
: Dave Sill, *The qmail Handbook,* Apress, 2002. ISBN: 1-893115-40-2.

web: email
: Wikipedia, *Email.* `http://en.wikipedia.org/wiki/Email`

web: gsoc
: *Google Summer of Code.* `http://code.google.com/soc/`

web: lbl
: Craig Leres, *LBL changes.* `http://savannah.nongnu.org/people/resume.php?user_id=20830`

web: nmh-workers
: Website of *nmh-workers mailing list.* `https://lists.nongnu.org/mailman/listinfo/nmh-workers`. Alternative mailing list archive: `http://www.mhonarc.org/archive/html/nmh-workers/`

web: rickert
: Website of *Neil Rickert's mh (nmh) scripts.* `http://faculty.cs.niu.edu/~rickert/mh/`

web: sloc-dwm
dwm: Lines of Code Through Time, Ohloh. http://www.ohloh.net/p/dwm/analyses/latest/languages_summary

web: sloc-wmii
wmii: Lines of Code Through Time, Ohloh. http://www.ohloh.net/p/wmii/analyses/latest/languages_summary

Wolter04
Jan Wolter, "DBM Hash Libraries," in *Unix Incompatibility Notes*, 2000–2004. http://www.unixpapa.com/incnote/dbm.html

XCU92
"Commands and Utilities (XCU), Issue 4," in *CAE Specification*, The Open Group, July 1992. ISBN: 1-872630-48-0.

XVS87
"XVS Commands and Utilities," in *X/Open Portability Guide*, vol. 1, January 1987. ISBN: 0-444-70174-5.

REQUESTS FOR COMMENTS

RFC 821
Simple Mail Transfer Protocol, August 1982.

RFC 822
Standard for the Format of ARPA Internet Text Messages, August 1982.

RFC 934
Proposed Standard for Message Encapsulation, January 1985.

RFC 1864
The Content-MD5 Header Field, October 1995.

RFC 2045
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996.

RFC 2046
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996.

RFC 2047
MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, November 1996.

RFC 2048
Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, November 1996.

RFC 2049
Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples, November 1996.

RFC 2822
Internet Message Format, April 2001.

RFC 3156
MIME Security with OpenPGP, August 2001.

RFC 4880
OpenPGP Message Format, November 2007.

WEBSITES OF SOFTWARE PROJECTS

dma	`https://github.com/corecode/dma`
dwm	`http://dwm.suckless.org`
fdm	`http://fdm.sourceforge.net`
fetchmail	`http://www.fetchmail.info`
getmail	`http://pyropus.ca/software/getmail/`
git	`http://git-scm.com`
gnupg	`http://www.gnupg.org`
masqmail	`http://marmaro.de/prog/masqmail/`
midnight commander	`http://www.gnu.org/software/mc/`
MH	`http://rand-mh.sourceforge.net`
MH-E	`http://mh-e.sourceforge.net`
mmh	`http://marmaro.de/prog/mmh/`
mpop	`http://mpop.sourceforge.net`
mutt	`http://www.mutt.org`
nmh	`http://nmh.nongnu.org`
nullmailer	`http://untroubled.org/nullmailer/`
Postfix	`http://www.postfix.org`
procmail	`http://www.procmail.org`
qmail	`http://cr.yp.to/qmail.html`
Sendmail	`http://www.sendmail.com`
sloccount	`http://www.dwheeler.com/sloccount/`
ssmtp	`http://packages.qa.debian.org/s/ssmtp.html`
wmii	`http://wmii.suckless.org`

CONFIRMATION

I hereby affirm that this thesis is the result of my own work, except where otherwise indicated.

Breitingen, 2012-07-16

Markus Schnalke, #693913

COLOPHON

This document was typeset with the *troff* document preparation system on Unix. After having typeset my diploma thesis with LaTeX, the choice for troff was similar to preferring MH over mutt.

I used the troff implementation of the Heirloom doctools, and built upon the *ms* macro package. To meet my personal wishes, I added further macros and replaced clumsy parts of ms. My own macro code comprises about 400 lines. Unfortunately, I must admit that the troff sources are not perfectly portable as I accessed Heirloom troff extensions and ms internals, occasionally. The typesetting command line read something like:

```
export TROFFONTS=fonts REFER=bib
soelim style *.roff | \
     refer -e -P -sLAD -1,2 -k | tbl | grap | pic | \
     troff -Tps -ms -mpictures 2>err.ig | dpost >thesis.ps
```

My document preparation setup was inspired and guided by Dougherty and O'Reilly's *UNIX Text Processing* and by chapter seven of Bourne's *The UNIX System*. The *Nroff/Troff User's Manual* helped with definitive answers.

The mail agent diagram was written in *pic*. The figures displaying the number of switches and the amount of code through time were created with *grap*. The build dependency graph was generated with *dot*. Source data was preprocessed with *awk*. For programming and for writing this document, I used the *ex-vi* editor because *ed* would have been a bit too heavy, even for me. ;-)

The text and heading font is Philipp Poll's *Linux Libertine*. The monospace font used for code listings is a TrueType variant of Dimitar Zhekov's *Terminus* font. Both are free typefaces.

The layout of the inner pages of this document is modeled after the German book *Einführung in die Automatentheorie, Formale Sprache und Komplexitätstheorie* by Hopcroft and Ullman, Addison-Wesley, 1990. The title page was inspired by books of the 19th century, mainly by the ones of Charles Darwin.

The complete sources of this document, as well as final versions in PDF and Postscript format, are available on my website: `http://marmaro.de/docs`.

Dijkstra's words on page iii are a quotation of EWD 648.