Mahogany Hacker's Guide
for Version 0.23a
"Polwarth III"

Written by Karsten Ballder and Vadim Zeitlin
<mahogany-developers@lists.sourceforge.net>

April 19, 2001

Contents

Chapter 1

Compiling *Mahogany* from source

These compilation notes are probably a bit outdated. The best start is to use the `--help` option of configure to see which options it supports. These notes are written with the Unix version in mind, for problems with compiling the MS Windows or MacOS versions, you might want to contact the mahogany-developers@lists.sourceforge.net mailing list instead.

1.1 General Procedure

1.1.1 Prerequisites

Before you can compile Mahogany, you will need a matching copy of wxWindows[1] .As wxWindows/wxGTK is under constant development, you should use the same copy of wxGTK that we used to build Mahogany, you can get it from the Mahogany download page[2].

1.1.2 Configuring wxGTK

Unpack wxGTK and run its `configure` script:

> wxGTK's configure system is a bit complicated, but easy to use once you see how to. If you are using the sources from the CVS

[1] http://www.wxwindows.org/

[2] http://mahogany.sourceforge.net/download.html

repository or the snapshot from the Mahogany ftp server, please follow the instructions in the file wxWindows/BuildCVS.txt. Generally this involves setting up a subdirectory for building it and invoking the configure script:

```
mkdir build
```

```
cd build
```

```
../configure --without-threads --with-gtk --without-shared
--without-debug_flag
--without-debug_info --without-odbc
```

```
make
```

```
make install
```

If compiling with gtk 1.0.x rather than 1.2.x, you need to specify the additional option `--without-dnd`. Of course, feel free to change the two debug options to `--with-debug` if you want to generate a version with debugging enabled. Just make sure that if you configure wxGTK with debug, then you also need to configure Mahogany `--with-debug`, or it will fail. If you experience problems with thread support, configure it `--without-threads`, they are not used currently. *Note: the –with/–without arguments are prefixed with a double hyphen, not just a single one, as it might show up in some version of this document.*

More up-to-date information can be found in Mahogany's README file.

1.1.3 Configuring Mahogany

This is even easier than with wxGTK. Just run Mahogany's `configure` script. If you have built wxGTK with debugging enabled, specify the additional option `--with-debug` when configuring Mahogany. Call `./configure --help` to get a list of supported options. Generally you will just run it without options. If configure fails to find some header files or libraries, you can create links to them in the `extra/include` and `extra/lib` directories and configure will find them. To reconfigure Mahogany, make sure that you delete any `*cache*` files first. If you have multiple versions of wxGTK installed, you can set the `WX_CONFIG` environment variable to tell `configure`

which `wx-config` program to call. *Note: the –with/–without arguments are prefixed with a double hyphen, not just a single one, as it might show up in some version of this document*

What to do, step by step:

```
./configure --help
./configure with options as described above
make depend
make
make install
make install_all
```

1.1.4 Compiling Mahogany

Just run `make` from the top level Mahogany directory. If you later want to rebuild Mahogany without rebuilding the libraries contained in the `extra/src` directory, i.e. after changing the Mahogany source, just run `make` in the `src` directory.

To generate the documentation

...type `make doc` from the main directory or simply `make` in the doc directory.

1.1.5 Installing Mahogany

Just call any or all of the following make targets in the main directory:

make install

Will compile and install Mahogany and the required additional files.

make install_doc

Will install and, if necessary, regenerate the documentation and online help system.

make install_locale

Will attempt to generate message catalogs for translations and install them. Note that Mahogany is not fully translated yet.

make install_all

Will do all of the above

1.1.6 Some extra software might be required

For regenerating the documentation you will need `doc++` for generating the class documentation. Alternatively the Makefile system will try to use kdoc or scandoc (included), but might fail halfway during class documentation generation. Also, latex, latex2html, netpbm and plenty of disk space on `/tmp` are required for generating the online docs. However, the distribution includes a file Mdoc.tar.gz in the doc directory. As long as you leave this file there, the Makefile will not attempt to regenerate the documentation but use this instead.

1.2 Operating systems specific

1.2.1 Linux

If compiling with a non-default compiler like `egcs`, make sure that `/usr/include` is not in the include path, neither should `/usr/lib` be explicitly listed. Mahogany has been compiled with `egcs` and `gcc-2.8.x` on both, `libc5` and `glibc2` systems.

1.2.2 Solaris/SunOS

Mahogany has been successfully compiled with `gcc-2.8.0` on Solaris. Currently it does not compile with the standard `C++` compiler.

1.2.3 Microsoft Windows

Mahogany can be compiled under Windows, using wxWindows Version 2.0 and Microsoft Visual C++ version 4.x or later. The makefiles are included in the Mahogany source distribution (they will build the version of Mahogany without Python support).

1. Prerequisites:

(a) You need the Mahogany sources. You may either download them from one of the locations mentioned in README file or use the cvs (preferred because it will allow us to fix reported bugs and for you to test our fixes faster)

(b) You need the sources of wxWindows cross-platform GUI library used by Mahogany: get it from either Mahogany home page or directly from http://www.wxwindows.org.

(c) You need a compiler capable of creating Win32 executables. Although, in principle, any one will do, we only provide makefiles for Microsoft Visual C++ - for other compilers you would need to create them yourself.

(d) The *source* distribution of Python 1.5 if you want to build Mahogany with Python support (if you don't know what Python is, you don't need it).

(e) You need quite a lot of time: on a slow machine (average CPU, IDE disk subsystem) the compilation might take a long time (on the order of 30 minutes - one hour counting with wxWindows)

2. Building wxWindows:
This is documented in wxWindows documentation. The makefiles and/or project files are provided for all major compilers (including Microsoft, Borland, Symantec, Watcom and g++ for Win32 (mingw32)). You will need almost all features of the library, so don't edit setup.h and leave the default settings in it. The important moment is to realize that the program should be built later with the same compile-time options and compiler settings as the library - if they are not identical, weird and difficult to diagnose problems will result! It is recommended (although not mandatory) that you use project files and not the command-line makefiles to build wxWindows if you are using Visual C++ compiler. Although wxWindows itself doesn't need it, you should define an environment variable WX with the path to the root directory of wxWindows installation (for example: set wx=d:\progs\libs\wxWindows).

3. Building Mahogany with Visual C++ 6.0
Open the file M.dsw in MSDEV.EXE (IDE). This is the main workspace file for Mahogany and it contains several projects (you may get warnings about some projects being missing - don't worry):

(a) wxWindows project: you probably don't have it in the same place, so just ignore the warnings and either leave it unloaded or delete it from the workspace if you wish. You may also modify the path to the project file (wxWindows.dsp) to point to the project file you used to build wxWindows if this is how you built it - then you will profit from automatic dependencies between projects.

(b) C-Client project: this is the library which is included in Mahogany distribution and which must be built as well. Load the project (right click it with the mouse, choose "Load") and it will be done automatically the first time you build Mahogany.

(c) compface is the library for the X-Face support: follow the same instructions as above.

(d) MModule project: this project contains the loadable modules for Mahogany ("plug-ins"). They are in experimental state right now and you may compile Mahogany without them for now.

(e) python15 project: if you have installed source distribution of Python, you should have this file in it. When you will try loading the project, you will have a possibility to browse for puthon15.dsp file - select the one from the Python distribution. If you don't have Python, leave it unloaded.

(f) Any other projects in the workspace shouldn't be used and should be left unloaded.

(g) After loading the ones which you need (C-Client is mandatory, all the other optional), you may start the build. It should succeed :-) If it doesn't, please report the build log with the explanation of what happened to mahogany-developers@lists.sourceforge.net mailing list.

(h) Notice that the environment variable WX must be set before you launch MSDEV.EXE (otherwise it won't inherit it)!

4. Building Mahogany with Visual C++ 5.0
   Edit the workspace file wxWindows.dsw with a text editor and replace the string "Format Version 6.00" with "Format Version 5.00". Then proceed as above.

5. Building Mahogany with other compilers
   It should be possible to do it, but I don't have all Windows compilers

installed on my system, so I cannot guarante it. Most probably, you might need some minor tweaks, but almost surely no major changes will be needed.
The list of all Mahogany source files can be found by looking at the Makefile files in each subdirectory of src: SRC variable there contains all the source files in that directory.
Add all of the sources to the makefile/project file and also add the resource file res/M.rc to the project, choose the same options as for wxWindows compilation and see what happens. If it doesn't work, send you questions to the usual place (mahogany-developers@lists.sourceforge.net). If it does work, please tell us about it as well so that this file could be updated.

1.2.4 DEC Alpha with Digital UNIX 3.2c

(Karsten Ballder: It seemed at that time that the segmentation fault was caused by a bug in wxGTK. By now this should be fixed. However, no new attempt to compile Mahogany on DEC Unix has been made yet. However, I include Holger's report as a help for some future attempt.)

This section has been contributed by Holger Bauer `bauer@itsm.uni-stuttgart.de`[3] `http://www.itsm.uni-stuttgart.de/ bauer`[4]

1. First I took the following arguments for configure:
   `configure --x-includes=/usr/X11R6/include --x-libraries=/usr/X11R6/shlib --with-wxGTK --without-python`
   This would not work since `make dep` still tries to find `Python.h`.
   Therefore I created a dummy include file Python.h in `.../M/src/Python`.
   This helped to complete make dep.

2. However step 1 still reported some warnings:
   `.../M/extra/src/c-client/mail.h` would complain about a missing `linkage.h` header file. Also, a missing `osdep.h` file was reported.
   `.../M/extra/src/c-client/osdep.h` falsely pointed to `os_gof.h` but this file was missing. Therefore copied `os_osf.h` to `os_gof.h`. Also, copied `os_osf.c` to `os_gof.c` since `osdepbas.c` pointed to `os_gof.c`.

[3] bauer@itsm.uni-stuttgart.de
[4] http://www.itsm.uni-stuttgart.de/ bauer

3. `cd src;`
   `make;`
   This would fail since swig was not searched for by the configure script. Since even I gave –without-python to configure make would search in /src/Python and try to compile things there I decided to install python and swig.

4. The Python-FAQ states that python definitely has to be compiled with the DEC cc.

5. After that I took:
   `configure --x-includes=/usr/X11R6/include --x-libraries=/usr/X11R6/shlib --with-wxGTK`
   The make process would then stop in `.../M/src/adb/AdbFrame.cpp`.
   The line `AdbTreeElement *current = wxIsPathSeparator(strEntry[0u])...` would not compile since the wxString function takes a long integer (due to the 64 bit architecture).
   This error might disappear if somebody adjusts the wxString library accordingly.
   In meantime (for my local `libwx_gtk.so`) I changed to `strEntry[0]`.

6. Then only final linkage errors occured:
   `-lpthread` is not known, needs `-lpthreads`
   `-lcrypt` is not known, just omit
   `-ldmalloc` is not known, pass `--without-dmalloc` to `configure`.

7. Because the link process did not give me any error I expected Mahogany to run (at least some kind of unstable). But a core dump appeared immediately.
   `dbx` gives to following information:
   ```
   (dbx) run
   signal Segmentation fault at >*[mig_get_reply_port, 0x3ff83004a0c]
   stq r26, 0(sp)
   (dbx) where
   > 0 mig_get_reply_port(0xffffffffffffffff, 0xffffffffffffffff,
   0xffffffffffffffff, 0xffffffffffffffff, 0xffffffffffffffff)
   [0x3ff83004a0c]
   ```
   The output of `gdb` is not very appealing:
   `(gdb) run Starting program:  /usr3/bauer/src/M/M/src/M`

```
Program received signal SIGSEGV,
Segmentation fault.  0x3ff83008f5c in port_allocate ()
(gdb) where
#0 0x3ff83008f5c in port_allocate () #1 0x3ff83004a38 in mig_get_reply_port
()
#2 0x3ff83008f94 in port_allocate () #3 0x3ff83004a38 in mig_get_reply_port
()
```

This output runs until infinity ....

1.3 Configuration and Testing

Under Unix all configuration settings are stored in `~/.M/config` under Windows in the registry. To get an overview over all possible configuration options and their default values, set the value `RecordDefaults=1`. Under Unix, do this by creating a new `~/.M/config` file containig the lines

```
[M]
RecordDefaults=1
```

After running Mahogany, this file will then contain all default settings. Most of them are easily understood. Otherwise, the file `include/Mdefaults.h` contains them all with some short comments.

1.4 wxWindows

You will need the wxWindows GUI toolkit to compile *Mahogany*.

Further Information

- wxWindows is available from `http://www.wxwindows.org/`
- The GTK port of wxWindows, wxGTK, is available from: http://www.wxwindows.org/dl_gtk

In order to compile this alpha release of *Mahogany* you will need wxWindows 2.0 strictly later than beta 3 and so it is probably easier to take the wxWindows sources snapshot from the *Mahogany* home page.

Chapter 2

The Real Hackers' Guide

Here we are trying to provide some information for existing and potential *Mahogany* developers.

2.1 Profiles

2.1.1 Description and Motivation

Profiles are objects storing configuration information. The purpose of using profiles rather than using the normal wxConfig system is to have *inheritance*. Each profile relates to some real existing object in Mahogany, for example, a mailfolder "MyFolder" would have a FolderProfile with the same name as the folder associated with it, or the settings of the FolderView window would read their settings from a profile with the name "FolderView". So each object requests a profile of well-defined name relating to the object, to access its configuration information. So far this is not different from sections or groups in a config file.

2.1.2 Inheritance

Whenever a profile is being created, it takes the pointer to a parent profile as argument in the constructor. This pointer may be NULL. A profile is therefore defined by two parameters: its name and its parent profile.

The Motivation behind this is to have the behaviour of different parts of Mahogany change dynamically as they are being used, to make them context aware. For example, a ComposeView used to reply to a message from a

mailing list might set a different return address to a ComposeView writing a message to a friend. Depending on from where the ComposeView was created, it will have different parent profiles (i.e. the profile of the folder with mailing list mails or the INBOX mailfolder profile or the mailfolder containing mails from my friends). Also, the MessageView window can inherit its settings from the folder that its messages are stored in (it inherits from the FolderView which inherits from the mailfolder) and therefore if a mailfolder is likely to contain lots of long mails (e.g. the Mahogany developers ranting about the advantages and shortcomings of different features), it can have a smaller typeface configured for displaying the messages – configured at the mailfolder level, not for a particular MessageView.

2.1.3 Which Profiles are there?

A list of all profiles can be obtained with `static kbStringList *ProfileBase::ListProfiles(ir type = PT_Any)`, which will return a list of either all profiles or those having the specified type if it is not `PT_Any`.

2.1.4 Technical description

Initialisation

When a profile is created, it will check whether a profile file with its name exists. For example MyFolder.profile or ComposeView.profile. If so, the profile will create a wxFileConfig object representing this file, otherwise this pointer remains NULL.

Reading values

When values are read, the profile will first check its own configuration, then try to inherit the value if it doesn't find it. It goes through the following steps, returning a value as soon as one is found:

1. If its own wxFileConfig *m_config is not NULL, it will try to read the value from there (i.e. from the profile file).

2. It will look up its *exact location* in the global configuration object. The *exact location* is found by asking its parent profile for its path and appending its own name to it. I.e. the ComposeView profile called from

the main window would return an *exact location* "M/Profiles/ComposeView". If a ComposeView window was opened from a FolderViewFrame showing a mailfolder "MyFolder", the path might look like this: "M/Folders/MyFolder/ComposeVi as the profile was created with the mailfolder profile as parent. So it would try to read the value from this section if it exists.

3. If still not found, it will go up the path in the configuration and try to find the value there, i.e. it would try "M/Folders/MyFolder", "M/Folders" and "M".

4. If not found, it will forward the request to its parent profile.

5. If not found, it will look in the global profile (==global configuration in [M/Profiles] section).

6. otherwise return the default value

Writing values

If the profile has its own wxFileConfig object representing a configuration file, then values will be stored there, otherwise in the *exact location* in the global configuration.

Under no circumstances will any values be modified in the parent profiles.

2.1.5 Shortcomings and things to fix

- Names of profile files should be case insensitive.
- Check and verify all locations of profile creation for sensible use of parent profiles (i.e. inherit from mailfolder profile but not from folderviewprofile).
- remove redundancy in lookup routines, i.e. going up in path is identical to inheriting.
- What is desperately needed is some more GUI support for this.I envision a profile editor showing a list or treectrl of profiles from which to pick, combined with a notebook of all available profile settings, whether they belong to the object in question or not. Also it would be nice if

all preferences pages had a little boolean checkbox to activate or deactivate them, so one can explicitly say "don't set this value, inherit it" so the value would be removed from the profile in question.

- Profile interface needs a function to remove a value.
- I have to add a function to retrieve a list of all Profiles and of all Profiles of a given type (e.g. folder profiles).
- All access to wxConfig pointers in profiles and groups etc should be removed. Exception: the global appplication config profile which needs to make its wxConfig pointer available for use by the wxPersistentControl classes.(though these would be much more useful if they took profile pointers instead, but I don't insist for now for the sake of reusability of them).

2.2 Filtering

The following proposal is currently being implemented in source.

Filtering rules in Mahogany are described in a simple control language, interpreted by a recursive descent parser. No yacc or flex required. I try to explain this in some form of BNF notation:

RULE:

CONDITION EXPRESSION EXPRESSION

// if true // if false

CONDITION:

(CONDITION LOGICAL_OPERATOR CONDITION)

SIMPLE_CONDITION

EXPRESSION:

ACTION_COMMAND

RULE

SIMPLE_CONDITION:

match_exact(arg, flags)
match_substring(arg, flags)
match_regexp(arg, flags)
python_rule(pythonfunc, args)
...

ACTION_COMMAND:

move_to_folder()
delete()
adjust_score() // I'd like scoring of messages like in GNUS, useful for news
print()
forward_to()
send_reply()
python_acton(pythonfunc, argc)
...

I believe this would be:

- easy to parse by the program
- easy to understand and edit by humans as it is stored in a programming-language like style
- easily extensible
- powerful enough to build rather complex expressions and rules

A Filter object would be an abstract representation of a set of rules parsed from a string (can contain more than one rule). It might be used by applying it to a given Message object. A subset of the Filter functionality could be a MatchTest object, which only knows conditions but no rules. It would return true if its conditions all return true (includin possible logical operators of course), false otherwise. It is used by the Filter class and could be used by some other Mahogany functionality like e.g. a "vacation" function which needs to check for mail-loops. Though, all this can actually be achieved with Filters, there shouldn't really be a need for a MatchTest class.

2.3 Adressbook (ADB) classes

Tech documentation of address book related classes.

2.3.1 Introduction

terms

"Address book" will be abbreviated as "ADB" from now on.

The ADB classes stor and allow to retrieve/edit e-mail addresses indexed by aliases/nicknames and other related information. It's important that ADB records are hierarchally organized, i.e. there may be not only entries but also groups containing other entries and groups. There may be several address books, possibly in different formats.

FIXME: what happens if the same nick is found in several ADBs?

data stored

each ADB entry must have at least the nickname and at least one e-mail address. The fields are now supported (* means required):

- Nick Name (*)
- Full Name
- First Name
- Family Name
- Prefix (M/Mme/...)
- Title
- Organisation
- Birthday
- Comments (any text, multi-line)
- EMail (*)
- HomePage

- ICQ
- Prefers HTML (boolean, if not - send text only)
- Other EMails (list of additional e-mail addresses)
- Home and office address

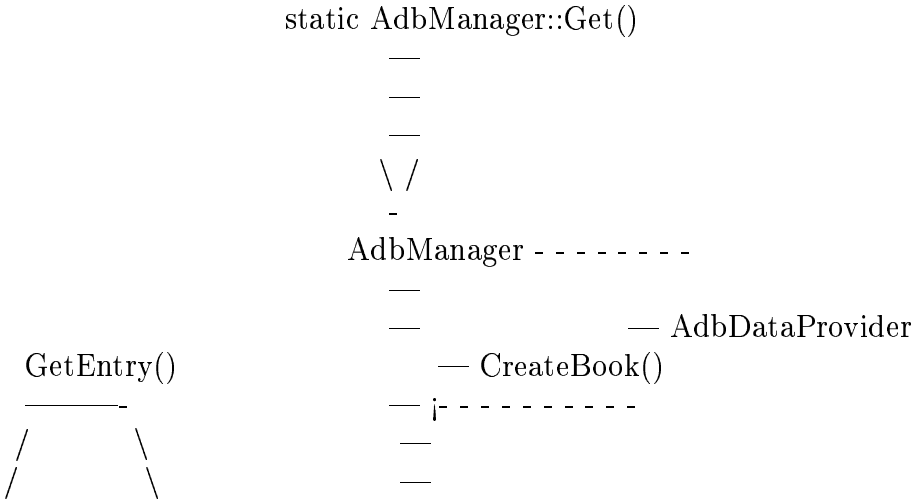
current ADB files format (Mahogany version 0.xx)

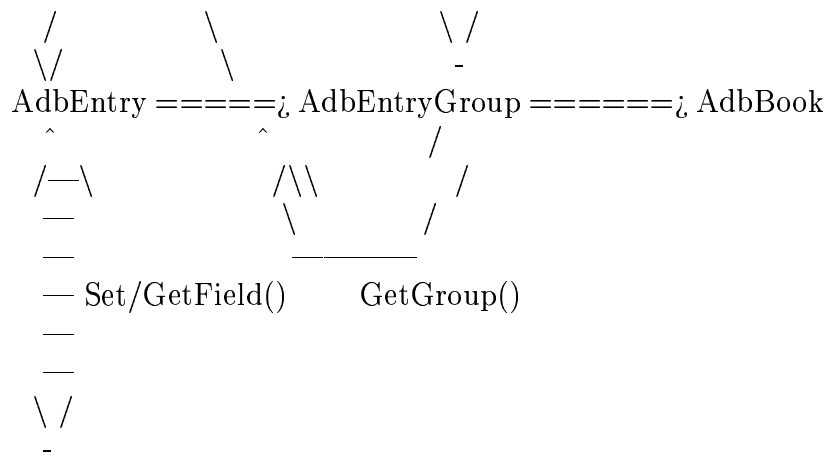
TODO

- Native Format using wxFileConfig
- BBDB Version 2, support for Emacs style .bbdb files

2.3.2 Classes

Class hierarchy is pictured below. Double arrows indicate inheritance, simple ones - functions which allow to get an object of this type. Dashed lines indicate "inner working" of the classes, i.e. something that you can safely ignore if you only want to use (and not modify) ADB code. All of these classes are the interfaces - i.e. they have no data and all their methods are public pure virtual (except for AdbManager, but it's almost like this...). All of the classes are reference counted (they derive from MObjectRC) and thus the usual rules of workign with ref counted objects apply (see MObject.h for details).





Brief description of each class (see corresponding header for more detailed descriptions and method prototypes):

AdbManager There is a single object of this class in the program.

It's responsible for managing all address books used by the application. It is created (transparently) with a call to AdbManager::Get(). It maintains a cache of all opened ADBs for efficiency and has methods to retrieve a pointer to AdbBook objects (which will be created using AdbDataProvider if they don't exist yet).

AdbDataProvider This is the class responsible for actually creating the address books objects. Each AdbDataProvider corresponds to one particular implementation of ADB, i.e. one provider is used for working with ADBs in Mahogany's native format, but another one may be used to work with ADBs in some other application's format or with remote (global wide) directory services accessible on the Internet.

TODO expand! this is really crucial...

AdbEntry Is just a record in an ADB. Contains all information corresponding to the given nickname and the methods to access it.

AdbEntryGroup Is a group of AdbEntries (no kidding) and has methods to retrieve/create/delete entries and subgroups.

AdbBook Is just a root AdbEntryGroup which has some additional attributes such as description.

Global functions

AdbManager::Get() returns the pointer to global (unique) AdbManager object, creating one if it doesn't exist. You must call Unget() exactly once for each call to Get().

AdbManager::Unget() decrements ref count of AdbManager, must be called for each (successful) call to Get().

::AdbLookup(...) looks in the specified address book(s) for the match for the given string. If the array of books is empty (or the pointer is NULL), all currently opened books are searched. All pointers returned in aEntries must be Unlock()ed by the caller (as usual).

Examples

TODO (this is probably the section Karsten is waiting the most for...) (Yes indeed :-)

2.4 MModules

MModules are DLLs which can be loaded into Mahogany at run time. This allows us to keep the main executable small and put all additional non-core functionality into add-on modules. The general structure of a module is very simple and can be understood by looking at Mdummy.cpp. It needs to contain a small number of functions with extern "C" linkage which are used to obtain information from the module and initialise it. Then, from the initialisation function, it can do anything it wants.

2.4.1 .MMD files

Each module must be accompanied by a .mmd file with the same name as the module itself. This file contains a few lines explaining the module's purpose and is used to present a dialog for configuring the modules. The lines are:

- "Mahogany-Module-Definition" ¡- a header ID to recognise valid files
- "Name: Mdummy" ¡- the symbolic name
- "Version: 0.123" ¡- the version as a string
- "Author: John Doe ¡john@somewhere.org¿" ¡- the author
- "" ¡- an empty line
- "multiple lines of text describing the module's functionality"

2.5 Multi-Threading, ASMailFolder and Locking